TEC-0114

Unmanned Ground Vehicle System Perception for Outdoor Navigation

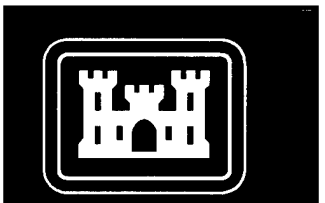
Charles Thorpe Martial Hebert
Dean Pomerleau Anthony Stentz
Takeo Kanade

Carnegie Mellon University
Robotics Institute
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

August 1998

Prepared for:
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Monitored by:
U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, VA 22315-3864

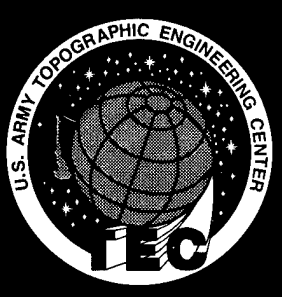


US Army Corps of Engineers
Topographic Engineering Center

T

E

C



Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1998	Technical August 1989-September 1995

4. TITLE AND SUBTITLE

Unmanned Ground Vehicle System Perception for Outdoor Navigation

5. FUNDING NUMBERS

DACA76-89-C-0014

6. AUTHOR(S)

Charles Thorpe Martial Hebert Dean Pomerleau
Anthony Stentz Takeo Kanade

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Carnegie Mellon University
Robotics Institute
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive, Arlington, VA 22203-1714

U.S. Army Topographic Engineering Center
7701 Telegraph Road, Alexandria, VA 22315-3864

19. SPONSORING / MONITORING AGENCY REPORT NUMBER

TEC-0114

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research includes perception for road following and cross-country navigation, path planning, and tools for building integrated mobile robot systems. Main accomplishments, software components, publications, and personnel are reviewed. Detailed technical descriptions of the key component of the research are provided.

14. SUBJECT TERMS

Unmanned Ground Vehicles, Sensing, Navigation, Obstacle Detection and Avoidance, Mobility, Autonomous Mobility

15. NUMBER OF PAGES

227

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

Table Of Contents

Chapter I: Introduction 1
Overview 1
The HMMWV Testbed 1
Key Results 2
Personnel 6
Technology Transfer 7
Publications 8

Chapter II: Architecture and Tools 13
Introduction 13
A Behavior-Based Architecture for Mobile Navigation 16
SAUSAGES: Between Planning and Action 25

Chapter III: Road Following 37
Introduction 37
Neural Network Vision for Robot Driving 39
Vision-Based Neural Network Road and Intersection Detection 53

Chapter IV: Obstacle Avoidance and Cross-Country Navigation 62
Introduction 62
SMARTY: Pixel-Based Range Processing for Autonomous Driving 64
RANGER: A Feedforward Control Approach to Autonomous Navigation 75
Sensor Fusion For Autonomous Navigation Using Neural Networks 101
A Practical Stereo Vision System 109
Sonar-Based Outdoor Vehicle Navigation 118
STRIPE: Low-Bandwidth or High-Latency Vehicle Teleoperation System 137

Chapter V: Path Planning 147
Introduction 147
D*: Optimal and Efficient Path Planning for Partially-Known Environments 148
Dynamic Mission Planning for Multiple Mobile Robots 159

Chapter VI: Systems 169
Introduction 169
Integrating Position Estimation and Perception for Autonomous Navigation 171
An Integrated System for Autonomous Off-Road Navigation 186
A Navigation System for Goal Acquisition in Unknown Environments 197

List Of Figures

Chapter I: Introduction 1

Figure 1: The HMMWV used as the main testbed in this program. 2

Chapter II: Architecture and Tools 13

A Behavior-Based Architecture for Mobile Navigation 16

Figure 1: Behaviors sending votes to arbiter. 17

Figure 2: Command fusion process. 18

Figure 3: Field of regard voting and arbitration; (a) Local map around the vehicle; (b) Field-of-regard polygons shaded according to the votes assigned by the field of regard behavior; (c) Votes for panning angles. 18

Figure 4: Arc evaluation in the *Obstacle Avoidance* behavior 19

Figure 5: Resampling of ALVINN output layer. 20

Figure 6: Using D* to evaluate distance from goal. 20

Figure 7: Evolutionary development in DAMN. 21

SAUSAGES: Between Planning and Action 25

Figure 1: A sample annotated map. 26

Figure 2: Navigating an intersection with an Annotated Map. 26

Figure 3: Using a road follower to update positions on a map. 27

Figure 4: A link of SAUSAGES. 28

Figure 5: A plan as a graph of links. 29

Figure 6: Concurrent links. 29

Figure 7: The UGV link class hierarchy. 30

Figure 8: Using Annotated Maps as SAUSAGES links. 31

Figure 9: Mixing geometric and cognitive links. 32

Figure 10: Separation of planning and execution. 33

Figure 11: SAUSAGES interacting with external planners and monitors. 33

Figure 12: A SAUSAGES system completely interleaves planning and execution. 33

Figure 13: The ultimate SAUSAGES intelligent agent. 34

Chapter III: Road Following 37

Neural Network Vision for Robot Driving 39

Figure 1: Neural network architecture for autonomous driving. 40

Figure 2: Schematic representation of training "on-the-fly". The network is shown images from the onboard sensor and trained to steer in the same direction as the driver. 40

Figure 3: The single original video image is shifted and rotated to create multiple training exemplars in which the vehicle appears to be at different locations relative to the road. 41

Figure 4: An aerial view of the vehicle at two different positions, with the corresponding sensor fields of view. To simulate the image transformation that would result from such a change in position and orientation of the vehicle, the overlap between the two field of view trapezoids is computed and used to direct resampling of the original image. 42

Figure 5: A schematic example of an original image, and a transformed image in which the vehicle appears one meter to the right of its initial position. The black region on the right of the transformed image corresponds to an unseen area in the original image. These pixels must be extrapolated from the information in the original image. 43

Figure 6: An aerial view (left) and image-based view (right) of the two techniques used to extrapolate the values for unknown pixels. See text for explanation. 43

Figure 7: Three reduced resolution images of a two-lane road with lines painted down the middle and right side. The left image is the original coming directly from the camera. The middle image was created by shifting the original image to make it appear the vehicle was situated one meter to the right of its original position using the first extrapolation technique described in the text. The right image shows the same shift of the original image, but using the more realistic extrapolation technique. 44

Figure 8: To calculate a network's steering error, the best fit gaussian is found to the network's output activation profile. The distance between the peak of the best fit gaussian and the position in the output vector representing the reference steering direction (in this case the person's steering direction) is calculated. This distance, measured in units or neurons between the two positions, is defined to be the network's steering error. 45

Figure 9: Illustration of the "pure pursuit" model of steering. 46

Figure 10: Vehicle displacement from the road center as the Navlab was driven by networks trained using three different techniques. 48

Figure 11: Video images taken on three of the road types ALVINN modules have been trained to handle. They are, from left to right, a single-lane dirt access road, a single-lane paved bicycle path, and a lined two-lane highway. 50

Vision-Based Neural Network Road and Intersection Detection 53

Figure 1: Virtual view for detecting a perpendicular road. 55

Figure 2: Input reconstruction metric for different virtual views. 56

Figure 3: Virtual images at different distances in front of the vehicle and corresponding IRRE values. 57

Figure 4: IRRE curves after translation to the road center. 58

Figure 5: A case in which vehicle heading and road orientation are not matched. 59

Figure 6: Definition of the second reference point P2. 59

Figure 7: Virtual view configuration for the detection of a Y intersection. 60

Chapter IV: Obstacle Avoidance and Cross-Country Navigation 62

SMARTY: Pixel-Based Range Processing for Autonomous Driving 64

Figure 1: Perception system for autonomous navigation. 64

Figure 2: (a) Traditional image/grid processing; (b) Pixel-based processing. 66

Figure 3: Updating the state of a cell; (a) slope and confidence as functions of number of pixels; (b) estimated elevation in cell as function of number of pixels. 68
Figure 4: Scanline-based processing from a range image 69
Figure 5: A local map (bottom) and the corresponding array of votes (top). 70
Figure 6: Transformations and their interrelations. 71
Figure 7: (a) Grid processing time per scanline; (b) local map processing time as function of the number of obstacle cells. 73
RANGER: A Feedforward Control Approach to Autonomous Navigation 75
Figure 1: Basic pure pursuit algorithm. 82
Figure 2: Adaptive pure pursuit. 82
Figure 3: Feedforward pure pursuit. 83
Figure 4: Performance of feedforward pure pursuit. 84
Figure 5: Feedforward simulator. 87
Figure 6: Reference points. 87
Figure 7: Propulsion plant model. 89
Figure 8: Propulsion servo. 89
Figure 9: Steering servo loop. 90
Figure 10: Wheel collision. 92
Figure 11: Body collision. 92
Figure 12: Static stability. 92
Figure 13: Hazard signals. 93
Figure 14: Merit servo loop. 94
Figure 15: Performance of speed planning. 95
Figure 16: Linear system block diagram. 96
Figure 17: State space model. 97
Figure 18: Control laws. 98
Sensor Fusion For Autonomous Navigation Using Neural Networks 101
Figure 1: Monolithic navigation network. 103
Figure 2: MAMMOTH network for navigation. 104
Figure 3: Map of the simulated world. 105
A Practical Stereo Vision System 109
Figure 1: Typical requirements for two stereo applications. 110
Figure 2: The Dante Robot. 110
Figure 3: Computation of the SSD (2-camera case). 111
Figure 4: Two examples of SSSD curves. 112

Figure 5: Front view of stereo camera jig. 113
Figure 6: Original right-hand image. 114
Figure 7: Computed range image. 115
Figure 8: Computed elevation map. 116
Figure 9: Times for a variety of stereo computations. 116
Sonar-Based Outdoor Vehicle Navigation 118
Figure 1: Sensor configuration. 119
Figure 2: Hardware architecture. 120
Figure 3: Local grid map. 121
Figure 4: Object transformation in local grid map. 122
Figure 5: Sensor to object transformation. 122
Figure 6: Velocity control. 124
Figure 7: Updating sequence for local grid map. 124
Figure 8: Arc evaluation for collision avoidance. 125
Figure 9: Precalculating arcs. 126
Figure 10: Searching for parking space: (a) Approaching gap, (b) Detecting gap, (c) Preparing vehicle for parking manoeuvre, (d) Typical street scene. 128
Figure 11: Path parameter transformation. 129
Figure 12: Removing outliers from least square fits: (a) Map display, (b) Typical corresponding scene. 130
Figure 13: Path tracking parameters. 132
Figure 14: Pure pursuit. 133
Figure 15: Parallel parking. 135
STRIPE: Low-Bandwidth or High-Latency Vehicle Teleoperation System 137
Figure 1: The basic STRIPE scenario. 138
Figure 2: FELICS flat-earth reprojection. 139
Figure 3: Flat earth reprojection. 140
Figure 4: Side view of a road and the corresponding image, with designated waypoints. The giant pinhole camera represents the camera on top of the vehicle. Points on the image plane of the giant camera represent the 2-D waypoints chosen by the operator. The dotted lines show where those image points project onto the actual terrain. The intersection of each of the dotted lines with the actual road is the "correct" location of the 3-D waypoint. 141
Figure 5: The 2-D waypoints are initially projected onto the vehicle's groundplane. The x's represent the location of the projected point. 141
Figure 6: The same waypoints are reprojected onto the vehicle's new groundplane. 142
Figure 7: The vehicle continues its reproject-and-drive procedure. The path behind the ve-

hicle has been divided into small planar regions. 142
Figure 8: An "easy" road. 143
Figure 9: A "difficult" road. 143
Figure 10: The current STRIPE operator interface. 144
Figure 11: Dashboard interface. 145
Figure 12: Compass interface. 145

Chapter V: Path Planning 147

D*: Optimal and Efficient Path Planning for Partially-Known Environments 148
Figure 1: Backpointers based on initial propagation. 152
Figure 2: LOWER states sweep into well. 152
Figure 3: Final backpointer configuration. 153
Figure 4: Path planning with a complete map. 154
Figure 5: Path planning with an optimistic map. 154
Figure 6: Path planning with a coarse-resolution map. 155
Figure 7: Typical environment for algorithm comparison. 157
Figure 8: Comparison of D* to optimal replanner. 157

Dynamic Mission Planning for Multiple Mobile Robots 159
Figure 1: System Data Flow 162
Figure 2: Simulation Step 1 164
Figure 3: Simulation Step 2 165
Figure 4: Simulation Step 3 165
Figure 5: Simulation Step 4 166
Figure 6: Advanced Scenario 166

Chapter VI: Systems 169

Integrating Position Estimation and Perception for Autonomous Navigation 171
Figure 1: Fourteen ERIM laser range finder images. Dark pixels are close to the sensor and light pixels are farther away. Corresponding objects are linked together across images. 175
Figure 2: Noise in distance traveled and its dependence on vehicle steering. Vertical axis is calibration of distance traveled. Horizontal axis is steering wheel position from left turn to right turn. 176
Figure 3: Wheel angle vs. steering wheel position. 177
Figure 4: Kalman filter equations. 181
Figure 5: Map of test area showing the road and trees. 182
Figure 6: Vehicle position and uncertainty during a run. 183
Figure 7: Effects of road updates on position uncertainty. 183

Figure 8: Errors caused by road updates during inaccurate driving through a turn..... 184

An Integrated System for Autonomous Off-Road Navigation 186

Figure 1: Architecture of the navigation system..... 187

Figure 2: View of terrain and approximate path..... 188

Figure 3: Exact path of vehicle; the obstacle regions are shown as black dots; the intermediate goal points are shown as small circles. 188

Figure 4: Detailed views of three sections of the loop of Figure 2 189

Figure 5: Main loop in Ganesha. 191

Figure 6: Set of possible turning radii in DAMN for the cross-country navigation system. 192

Figure 7: Example vehicle path around an obstacle. 193

Figure 8: Distribution of votes in the arbiter (thick line), obstacle avoidance (thin line) and heading (dotted line) behaviors..... 194

A Navigation System for Goal Acquisition in Unknown Environments 197

Figure 1: Example of an actual run..... 198

Figure 2: Navigation system diagram..... 199

Figure 3: Planning with a complete map. 201

Figure 4: Planning with an optimistic map cell expansion from SMARTY data..... 201

Figure 5: D* overview. 202

Figure 6: SMARTY overview. 204

Figure 7: (a) Local map; (b) Corresponding distribution of votes..... 205

Figure 8: Aerial view of slag heap test site..... 207

Figure 9: Driving into the cul-de-sac. 208

Figure 10: Discovering the cul-de-sac. 209

Figure 11: Exiting the cul-de-sac..... 210

Figure 12: Looping back toward the "gap". 210

Figure 13: Driving through and around the vegetated area. 211

Figure 14: Finding the access road that leads to the goal. 212

Figure 15: Driving the initial path to the goal. 213

Figure 16: Manually driving to a new Start point..... 213

Figure 17: Finding the goal for a second time. 214

Figure 18: Driving manually to a third start point. 214

Figure 19: Driving to the goal using prior map data. 215

Preface

The Robotics Institute of Carnegie Mellon University has been conducting work in the area of autonomous mobile robots under DARPA funding. The goals of this work are: to develop core perception technology for the navigation of Unmanned Ground Vehicles (UGV) in outdoor environment, to develop planning, architecture and systems-building tools for the integration of the core technology in unmanned systems, and to demonstrate autonomous systems for outdoor navigation.

Substantial progress has been achieved in all three areas. Core perception technology includes vision for road-following, as demonstrated in one of the first systems for autonomous road-following at highway speeds, and several approaches to obstacle avoidance, including sonar, laser range finder, and passive stereo. A comprehensive set of systems building tools have been developed, including tools for inter-process communications, command arbitration, and mission management. Perception techniques and architectural tools have been integrated into operational systems and demonstrated in realistic systems. Much of the technology developed at the Robotics Institute was transferred to and demonstrated in the context of the DARPA UGV Demo II program.

This report reviews the results obtained at the Robotics Institute from August 1990 to February 1996 on research sponsored by DARPA jointly under contracts "Perception for Outdoor Navigation," contract number DACA76-89-C-0014 monitored by the U.S. Army Topographic Engineering Center and "Unmanned Ground Vehicle System," contract number DAAE07-90-C-R059 monitored by TACOM. This report includes results obtained during extensions to DAAE07-90-C-R059 from September 1995 to February 1996. Parts of this work were also supported by a Hughes Fellowship, by a DARPA Research Assistantship in Parallel Processing administered by the University of Maryland, by a Draper Labs Post-Doctoral fellowship, and by NSF, under grant BCS-9120655, "Annotated Maps for Autonomous Vehicles."

Detailed descriptions of the technology developed under these programs are included in the report, along with detailed analysis of the results obtained in exercising several systems for autonomous navigation. The results of this work constitute a technology base for further development of unmanned systems.

Chapter I: Introduction

1. Overview

This report reviews the results obtained at the Robotics Institute of Carnegie Mellon University from August 1990 to February 1996 on research sponsored by DARPA jointly under contracts "Perception for Outdoor Navigation," contract number DACA76-89-C-0014 monitored by the U.S. Army Topographic Engineering Center and "Unmanned Ground Vehicle System," contract number DAAE07-90-C-R059 monitored by TACOM. This report includes results obtained during extensions to DAAE07-90-C-R059 from September 1995 to February 1996. Parts of this work were also supported by a Hughes Fellowship, by a DARPA Research Assistantship in Parallel Processing administered by the University of Maryland, by a Draper Labs Post-Doctoral fellowship, and by NSF, under grant BCS-9120655, "Annotated Maps for Autonomous Vehicles."

The three goals of this research are:

- To develop core perception technology for the navigation of Unmanned Ground Vehicles (UGV) in outdoor environment.
- To develop planning, architecture and systems-building tools for the integration of the core technology in unmanned systems.
- To demonstrate autonomous systems for outdoor navigation.

As part of the third item, the work included delivery of the components to Lockheed Martin for integration into the testbed of the DARPA UGV Demo II program and participation in the annual demonstrations for the Demo II program. Throughout this report, we indicate those components that were delivered to Lockheed Martin and included in one of the intermediate demonstrations of the UGV Demo II program.

The report is organized as follows. This chapter is an overview of the project. We briefly review our main results in Section 3., including a list of software modules. In Section 4., we provide a list of the personnel, including faculty and students, involved in this work. The work reviewed in this report has been disseminated in two ways: technology transfer to other institutions or projects, and publications in technical conferences and journals. Those two aspects are described in Sections 5. and 6., respectively.

Chapters II to V contain detailed descriptions of all the techniques developed in this program. Each chapter starts with an overview of the algorithms and their performance, followed by detailed technical descriptions of the most important components. We conclude the report in Chapter VI with a detailed description of three systems demonstrated at CMU; these systems integrate most of the techniques described in the preceding chapters.

2. The HMMWV Testbed

New technology has been developed and demonstrated for all aspects of perception outdoor navigation, e.g. road following and obstacle avoidance; we have developed the necessary tools for integration of this core technology into systems capable of carrying out complex missions, e.g., mission planning and control software. Finally, we have contributed most of the mobility software to the systems used in Demos A, B, and C of the Demo II program.

The main testbed for this work is an HMMWV provided by TACOM (Figure 1.) The vehicle was retrofitted to include:

- *Actuation*: Computer-controlled actuators on throttle, brake, and steering.
- *Computing*: Three Sparc workstations for general computing and additional boards for real-time control.
- *Sensing*: Stereo cameras for obstacle avoidance; a video camera for road following and teleoperation; a laser range finder for obstacle avoidance; a doppler radar for ground velocity sensing; and an integrated navigation system (inertial and GPS) for attitude and position sensing.
- *Communications*: Ethernet for communications between on-board computing components and cellular modems for communication with other vehicles or with a ground station.

A real-time controller running under VxWorks was developed for low-level control of the vehicle, i.e. speed and steering control, and for status monitoring. All the modules described in this report are based on this controller. Early work in this program was conducted using a retrofitted Chevy van with sensing and computing equipment on board. Unless noted otherwise, all the results presented in this report were developed and tested using those two vehicles.

Figure 1: The HMMWV used as the main testbed in this program.

3. Key Results

The key results from this work are in four main areas: architecture, road following, obstacle avoidance and cross-country navigation, and path planning. We give a brief overview of the results achieved in those areas in Sections 3.1. to 3.4. Detailed technical descriptions are provided in Chapter II to Chapter V. We provide a complete list of the software components developed under this program in Section 3.5.A brief description of the integrated systems and demonstrations that we carried out to validate these results concludes this section.

3.1. Architecture

A behavior-based approach to unmanned vehicle architectures was developed in which software modules with well-defined responsibilities, called behaviors, control the vehicle. Example of behaviors include road following and obstacle avoidance. When necessary, the behaviors communicate with each other through point-to-point network connections without using a central communication module. The approach is a reactive approach in that the behaviors issue commands at a high rate, as opposed to other approaches in which mobility components plan further ahead but generate commands at a lower rate.

To implement this approach, three types of tools were developed: a communication package which includes all the functionality necessary for communication between software modules; an arbitration system which combines steering, speed and sensor orientation commands produced by different behaviors into actual commands to the vehicle controller; and a mission planning and control environment which controls the execution of the behaviors based on a mission plan.

A key feature of this approach is its flexibility. For example, behaviors can be developed and tested independently before their integration into a larger system. Conversely, the algorithms used within a particular behavior may be modified without affecting the rest of the system. Finally, incremental development of navigation systems is facilitated since new behaviors can be added without modifying the existing system.

This approach to architecture has been used throughout this work. It is the basis for all the systems and components described in this report. It is also the approach used in the Demo II integrated systems. The communications, planning, and arbitration tools have been used for building systems with over twenty modules, including four different behaviors generating driving commands. The performance of an autonomous vehicle is only as good as the performance of the individual behaviors. The next three sections describe the core perception and planning technology developed for building those behaviors.

3.2. Road Following

The first mobility component is autonomous road following for which we developed a technique based on neural networks. In this approach, after initial training, the neural network generates steering and speed commands based on the location of the road in a video image. This technique was demonstrated at 55mph over hundreds of miles of highways of different types, as well as on difficult dirt roads with sharp curves and without salient road features. Through those experiments, we have shown that the technique is robust with respect to illumination, shadows, road geometry, and changes in road appearance. In addition to the experiments documented in Chapter III, this road-following technique was used in Demos A, B, and C of the UGV Demo II program.

In this first approach, the road-following module can handle only the type of road for which it has been initially trained. The technique was extended through the use of multiple neural networks, thus enabling the system to handle several different road types and to transition smoothly between roads. This capability was demonstrated by driving the vehicle in a network of roads of different types.

Another limitation of the original approach is that it relies on viewing the road from a fixed camera. This problem was addressed initially by developing an additional module which controls the orientation of the camera based on the output of the road following program and on the current trajectory of the vehicle. This first approach was demonstrated by driving the vehicle under autonomous control of a dirt road with sharp curves and intersections, verifying that the sensor control module did maintain the road at the center of the image.

The most recent addition to road following is a more general approach to the camera control in which virtual images of the road are generated based on the current image. Those images show the road as it would be seen from a virtual camera which may be placed at any arbitrary location. In this approach, changes in road appearance and geometry can be detected in advance, intersections can be properly handled, and road detection from an off-road position is possible.

3.3. Obstacle Avoidance and Cross-Country Navigation

The second class of behaviors addresses the problem of obstacle detection and avoidance, and cross-country navigation using range images. Several perception techniques were developed to address this problem. One approach converts the range data to three-dimensional points and identifies obstacles based on the local shape

of the terrain, i.e. slope, height variation, etc. The distribution of obstacles in the resulting map is used for determining the best steering arcs and for controlling the speed of the vehicle. A second approach first builds an elevation map of the terrain seen in range images and computes the degree of safety of the vehicle at different locations in the elevation map. The safest trajectories are selected in order to generate steering and speed commands.

Those two approaches are complementary in that the first approach can detect small discrete obstacles, while the second approach is better suited to cross-country environments. Behaviors implemented using both approaches were demonstrated in natural outdoor environments, driving the vehicle over long distances. The behaviors were demonstrated both in isolation and as part of complex systems as described in Chapter VI. In addition, our obstacle detection behaviors are the core of the cross-country system demonstrated at the Demo C of the UGV Demo II program. They will also be the basis of the cross-country system for the final Demo II system.

The obstacle avoidance and cross-country navigation behaviors described above use dense range data from laser range finders or stereo vision systems. Techniques for obstacle detection and avoidance using point sensors such as sonars were also developed. Those techniques were demonstrated using a five-sonar system mounted on the vehicle. This showed the potential of sonars as "soft bumpers" or obstacle mapping sensors.

The core obstacle detection behaviors use exclusively the shape of the terrain in order to evaluate the navigability of the terrain. In real driving, however, the type of the terrain, e.g., dirt vs. mud vs. grass, is a critical part in determining whether a piece of the terrain is navigable. Algorithms for determining the type of the terrain from video and range images were demonstrated. The approach is based on using a neural network which is trained to recognize different types of terrain in images. Although terrain typing is not yet fully integrated in the cross-country system, it will be a major component of future perception systems for unmanned vehicles.

In many applications of unmanned vehicles, however, it is desirable to control the vehicle remotely. A module for teleoperation in uneven terrain was developed. The originality of our approach is that it is intended for very slow communication links, as is typically the case in secure digital communications, and it explicitly takes into account the fact that the actual terrain is uneven, instead of using a flat terrain assumption. The teleoperation system was demonstrated on the HMMWV using cellular modems and simulated communication delays. The teleoperation system was also demonstrated on the Demo II vehicles as part of Demo A, B, and C of the UGV Demo II program.

3.4. Path Planning

The road-following and obstacle avoidance behaviors described above generate driving commands based exclusively on local information such as road location or obstacle map. Driving based on local information is often called "tactical planning." In a realistic scenario, however, the system also needs a component of "strategic planning" in which the vehicle is instructed, for example, to reach a given goal position. The system needs to compute a path from the current position to the goal. The problem with strategic path planning is that the information necessary for correctly computing the path is not always available and may be discovered by the vehicle's sensors only as the vehicle travels. Furthermore, planning over a potentially large area may be costly in practice.

To address these problems, a novel path planning technology was developed which allows fast update of the current path based on new sensor information. In this approach, the vehicle may start with an empty map and an optimistic path to the goal point. As new sensor data becomes available, the path is modified as the vehicle is moving to take into account potential new obstacles.

In addition to being supported by theoretical optimality and complexity results, this path planning technique is the first to enable dynamic path planning over large areas. This path planner was used over long missions, i.e., several kilometers. The path planner has also been ported to the DemoII platforms and has been demonstrated

as part of Demo C. Finally, the path planner may be used for multi-vehicle operation in which the map built by a first vehicle is used by a second vehicle for more efficient path planning. Multi-vehicle operation was also demonstrated as part of Demo C. Finally, we have extended the path planner itself to handle multiple vehicles and multiple goals.

3.5. Summary of Software Components

The software components are listed below by name along with brief explanations of their functions. Technical details for all those components are given in Chapter II to Chapter V. All those software components run on Unix or VxWorks platforms. Each of those software components has been tested on our testbed; most have been transferred to Lockheed Martin for integration in the Demo II system. The components are as follows:

- IPT: Toolkit for inter-process point-to-point communications between modules.
- DAMN: Arbitration system for combining recommendations for steering, speed and sensor orientation produced by different behaviors.
- SAUSAGES: Software environment for describing mission plans and for run-time monitoring of mission execution.
- ALVINN: Road following module based on neural network training on a single type of road.
- MANIAC: Road following for multiple types of roads by integrating the output of several ALVINN networks.
- RACOON: Real-time vehicle tracking from a video camera.
- PANACEA: Control of camera orientation based on ALVINN output and current vehicle trajectory.
- GANESHA: Local obstacle map management from sonars or other sensors.
- RANGER: Cross-country navigation using range data from stereo vision or laser range finder.
- SMARTY: Obstacle detection using range data from stereo vision or laser range finder.
- MAMMOTH: Multiple neural networks for terrain classification and obstacle detection.
- STRIPE: Teleoperation over low-bandwidth video link.
- D*: Dynamic route planner for computing the optimal trajectory from the current position of the vehicle and a goal point based on an obstacle map.

This report describes the state of the software components described above as developed through August 1995, as well as modifications included as part of an extension to January 1996. Those modifications include:

- SMARTY: Inclusion of filtering of false alarms in the obstacle map, and adjustement of the obstacle sensitivity as function of the distance to the sensor.
- D*: Dynamic adjustment of the bounds of the D* obstacle map, improvement in the focussed search in the D* algorithm, and improvements in memory usage.
- IPT: Improvements in memory usage.

- MAMMOTH: Refinement and validation of the algorithms for classification.

3.6. Integrated Systems and Demonstrations

All the components described above have been individually tested and demonstrated. However, it is important to demonstrate those capabilities in the context of larger systems because, ultimately, unmanned vehicles will be used for complex missions. Also, building the systems allows us to evaluate the quality of our architectural tools and to refine them. Finally, we were able to evaluate the performance of the behaviors in the context of large systems before delivering them for integration in the UGV Demo II program.

Three such systems are described in Chapter VI. They are: map-driven navigation on roads including explicit representation of sensor uncertainty; cross-country navigation over long distances using a laser range finder controlled by intermediate goal points; and dynamic exploration of unknown cross-country terrain over long distances with a single goal point. The expression "long distances" in the last two examples means "greater than one kilometer." By showing navigation over long distances in natural terrain, those systems demonstrate the robustness of the perception and path planning techniques. Those systems show that the architectural tools described in Section 3.1. are well suited for large-scale systems and complex missions.

4. Personnel

Faculty members: Martial Hebert, Takeo Kanade, Dean Pomerleau, Steve Shafer, Tony Stentz, Chuck Thorpe, William Whittaker.

Graduate Students: Listed below are the students supported under this program who either proposed or defended their Ph.D. thesis.

Theses:

Ian Lane Davis, "A Modular Neural Network Approach to Multi-Sensor Autonomous Navigation," January 1996.

Todd Jochem, "Vision Based Tactical Driving," January 1996.

Alonzo Kelly, "An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem," July 1995.

Karl Kluge, "YARF: A System for Adaptive Navigation of Structured City Roads," February 1993.

Douglas Reece, "Selective Perception for Robot Driving," May 1992.

Dean Pomerleau, "Neural Network Perception for Mobile Robot Guidance," February 1992.

Thesis Proposals:

Omead Amidi, "An Autonomous Vision-Guided Helicopter," expected 1996.

Barry Brumitt, "A Generalized Autonomous Mobile Mission Planning System," expected 1997.

R Coulter, "A Systemic, Control Theoretic Approach to the Engineering of Autonomous Navigation System Motion Controllers and Motion Planners," expected 1996.

Jay Gowdy, "Emergent Architectures: A Case Study of Outdoor Mobile Robots," expected 1996.

Jennifer Kay, "Remote Driving with Limited Image Data," expected 1996.

Dirk Langer, "An Integrated MMW Radar System for Outdoor Navigation," expected 1996.

Julio Rosenblatt, "DAMN: A Distributed Architecture for Mobile Navigation," expected 1996.

Rahul Sukthankar, "Situational Awareness for Driving in Traffic," expected 1996.

Other Personnel: The list below includes all support staff, technical staff, and additional students employed on a part-time or full-time basis during part of the duration of the contract. The list does not include part-time undergraduates and students on a work-study program.

Allen, Paul	Athanassiou, Charalambos	Balakumar, Purushothaman
Baluja, Shumeet	Baun, Gary	Beck, Robert
Brajovic, Vladimir	Brown, H. Benjamin	Christian, Daniel
Collier, David	Curry, Patricia	Denton, Jutta
Elm, Marie	Fedor, Christopher	Fissell, Catherine
Fitzpatrick, Kerien	Fong, Terrence	Frazier, James
Gross, Patrick	Hancock, John	Heiler, Michael
Kang, Sing Bing	Krawczykiewicz, Carolyn	Krumm, John
Livington, David	Marcus, Lori	Marsh, Florence
Martin, James	Meadows, Timothy	Moody, James
Motazed, Behnam	Mueller, George	Nolla, Anthony
Osborn, Barbara	Pahnos, David	Pobla, Alban
Porsche, Kathryn	Ross, William	Schempf, Hagen
Shi, Wenfan	Shin, Dong	Simon, David
Smith, Bryon	Taylor, C. Roy	Voyles, Richard
Williamson, Todd		

5. Technology Transfer

All the software developed in this program was transferred to Lockheed Martin for integration in the Demo II vehicles. In addition, many of the software components were transferred to other organizations both inside and outside the UGV community:

- SAUSAGES: Georgia Tech. and University of Texas at Austin use SAUSAGES as part of their planning work; Hughes AI used SAUSAGES in their mission planning environment; Florida Atlantic University and Draper Labs are using SAUSAGES for underwater vehicle applications.

- ALVINN: RedZone Inc. used the road following software as part of their convoying application; University of Massachussets integrated ALVINN into their autonomous vehicle system; ALVINN was also transferred to the Army Research Labs for evaluation.

- DAMN: Hughes AI and Georgia Tech. used DAMN in their simulation work.

- D*: D* is integrated in the Hughes STX user interface environment.

- SMARTY: The NIST/ARL vehicle was demonstrated using SMARTY with the Dornier EBK laser range finder.

- RANGER: RANGER was demonstrated on a JPL testbed using real-time stereo; RANGER was also transferred to NIST/ARL for possible integration in their system.

- GANESHA: JPL and University of Michigan used GANESHA for obstacle mapping and avoidance; GANESHA was used at Teleos for controlling a small robot equipped with sonar sensors.

- IPT: IPT (and its earlier versions) was transferred to several institutions, among them Florida Atlantic University, Draper Labs, Wright AFB.

Within Carnegie Mellon, the ideas developed under this program have been instrumental in other projects, for example:

- RANGER was used as the first generation mobility module for a Lunar Explorer using stereo vision.

- Expertise in stereo and laser range finder processing and evaluation were used in the development of the legged robots, Dante II and AMBLER.

- The initial work on ALVINN was the basis for the road following part of the Automated Highway Systems (AHS) project of the Department of Transportation. Concepts from the GANESHA system are also being used in the AHS project for radar-based obstacle avoidance.

6. Publications

[1] D. Aubert, K. Kluge, and C.E. Thorpe. Autonomous Navigation of Structured City Roads. In *Proceedings of SPIE Mobile Robots V.* 1990.

[2] B.L. Brumitt, A.J. Kelly, A. Stentz. Dynamic Trajectory Planning for a Cross-Country Navigator. *Proceedings of SPIE Mobile Robots VII.* Boston, MA. 1992.

[3] B.L. Brumitt, R. Coulter, A.J. Kelly, A. Stentz. A System for Cross Country Navigation. *Proceedings of IFAC International Symposium on Intelligent Components and Instruments for Control Applications (SICICA).* Malaga, Spain. 1992.

[4] J. Crisman and C.E. Thorpe. SCARF: A Color Vision System that Tracks Roads and Intersections. *IEEE Trans. on Robotics and Automation.* Vol 9 # 1. 1993.

[5] K. Fitzpatrick, M. Hebert, D.A. Pomerleau, H. Schempf. Autonomous Cargo Transport System. *Proceedings of the United States Postal Service 5th Advanced Technology Conference.* Washington, DC. 1992.

[6] J. Hancock and C.E. Thorpe. ELVIS: Eigenvectors for Land Vehicle Image System. *Proceedings IROS 95,* also CMU Technical Report CMU-RI-TR-94-43. 1994.

[7] M. Hebert. Pixel-Based Range Image Processing. *Proceedings IEEE International Conference on*

Robotics and Automation. 1994.

[8] M. Hebert. 3-D Landmark Recognition from Range Images. *Proceedings CVPR'92*. Champaign, Illinois. 1992.

[9] M. Hebert and E. Krotkov. 3D Measurements for Imaging Laser Radars. *Image and Vision Computing 10:3*. 1992.

[10] M. Hebert. Building Object and Terrain Representation for an Autonomous Vehicle. *Proceedings American Control Conference*. June 1991.

[11] I.L. Davis and A. Stentz. Sensor Fusion For Autonomous Outdoor Navigation Using Neural Networks. *Proceedings IEEE Conference on Intelligent Robots and Systems*. Pittsburgh, Pennsylvania. 1995.

[12] T.M. Jochem and S. Baluja. Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following. In *Massively Parallel Artificial Intelligence*. Hiroaki Kitano, (Ed.). AAAI Publishers in cooperation with MIT Press. 1993.

[13] T.M. Jochem and S. Baluja. Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following. Technical Report. CMU-RI-TR-93-10. Pittsburgh, Pennsylvania. 1993.

[14] T.M. Jochem, D.A. Pomerleau, and C.E. Thorpe. Vision-Based Neural Network Road and Intersection Detection and Traversal. *Proceedings IEEE Conference on Intelligent Robots and Systems*. Pittsburgh, Pennsylvania. 1995.

[15] T.M. Jochem, D.A. Pomerleau,and C.E. Thorpe. MANIAC: A Next Generation Neurally Based Autonomous Road Follower. *Proceedings of the International Conference on Intelligent Autonomous Systems*. C.E. Thorpe (ed.). IOS Publishers, Amsterdam. 1993.

[16] J.S. Kay, and C.E. Thorpe. Supervised Telerobotics Using Incremental Flat-Earth Geometry: STRIFE. In *Proceedings of Intelligent Autonomous Systems-3*. Pittsburgh. 1993.

[17] J.S. Kay. STRIPE: Remote Driving Using Limited Image Data. In *Proceedings CHI '95 Conference on Human Factors in Computing Systems*. Denver. 1995.

[18] J.S. Kay, and C.E. Thorpe. Operator Interface Design Issues In A Low-Bandwidth And High-Latency Vehicle Teleoperation System. *Proceedings 25th International Conference on Environmental Systems*. San Diego. 1995.

[19] A.J. Kelly. *Essential Kinematics for Autonomous Vehicles*. Robotics Institute Technical Report. CMU-RI-14-94. 1994.

[20] A.J. Kelly. *Modern Inertial and Satellite Navigation Systems*. Robotics Institute Technical Report. CMU-RI-15-94. 1994.

[21] A.J. Kelly. *A Partial Analysis of the High Speed Autonomous Navigation Problem*. Robotics Institute Technical Report. CMU-RI-16-94. 1994.

[22] A.J. Kelly. *A Feedforward Control Approach to the Local Navigation Problem for Autonomous Vehicles*. Robotics Institute Technical Report. CMU-RI-17-94. 1994.

[23] A.J. Kelly. *Adaptive Perception for Autonomous Vehicles*. Robotics Institute Technical Report. CMU-RI-18-94. 1994.

[24] A.J. Kelly. *A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles*. Robotics Institute Technical Report. CMU-RI-19-94. 1994.

[25] A.J. Kelly. *An Intelligent Predictive Controller for Autonomous Vehicles*. Robotics Institute Technical Report. CMU-RI-14-94. 1994.

[26] A.J. Kelly, A. Stentz, M. Hebert. Terrain Map Building for Fast Navigation on Rugged Outdoor Terrain. *Proceedings of SPIE Symposium on Mobile Robots*. 1992.

[27] K. Kluge. *YARF: A System for Adaptive Navigation of Structured City Roads*. PhD Thesis. CMU School of Computer Science. 1993.

[28] K. Kluge and C.E. Thorpe. Intersection Detection in the YARF Road Following System. In Proceedings *Proceedings of Intelligent Autonomous Systems-3*. Pittsburgh. 1993.

[29] K. Kluge and C.E. Thorpe. Representation and Recovery of Road Geometry in YARF. *Proceedings Intelligent Vehicles*. I. Masaki ed. 1992.

[30] K. Kluge. Representation and Recovery of Road Geometry in YARF. In *Proceedings Intelligent Vehicles*. I. Masaki ed. 1992.

[31] D. Langer, C.E. Thorpe. Range Sensor Based Outdoor Vehicle Navigation, Collision Avoidance and Parallel Parking. *Autonomous Robots*. Vol.2, No.2. 1995.

[32] D. Langer, J. Rosenblatt, and M. Hebert. An Integrated System for Autonomous Off-Road Navigation. *Proceedings IEEE International Conference on Robotics and Automation*. 1994.

[33] D. Langer, J.K. Rosenblatt, M. Hebert. A Behavior-Based System for Off-Road Navigation. *IEEE Transactions on Robotics and Automation*. Vol 10, No. 6. 1994.

[34] D. Langer, C.E. Thorpe. Sonar based Outdoor Vehicle Navigation and Collision Avoidance. *Proceedings IROS'92*. Raleigh, NC. 1992.

[35] D. Langer and C.E. Thorpe. Sonar based Outdoor Vehicle Navigation and Collision Avoidance. *Proceedings IROS'92*. Raleigh, NC. 1992.

[36] D.A. Pomerleau, C.E. Thorpe, D. Langer, J.K. Rosenblatt, R. Sukthankar. AVCS Research at Carnegie Mellon University. In *Proceedings Intelligent Vehicle Highway Systems*. 1994.

[37] D.A. Pomerleau. *Neural network perception for mobile robot guidance*. Kluwer Academic Publishing, 1993.

[38] D.A. Pomerleau. Reliability Estimation for Neural Network Based Autonomous Driving. *Robotics and Autonomous Systems* 12. 1994.

[39] D.A. Pomerleau. Neural Networks for Intelligent Vehicles. *Proceedings Intelligent Vehicles Conference*. I. Masaki (ed.). 1993.

[40] D. Pomerleau, C. Thorpe, D. Langer, J.K. Rosenblatt, R. Sukthankar. AVCS Research at Carnegie Mellon University. In *Proceedings of Intelligent Vehicle Highway Systems*. 1994.

[41] D.A. Pomerleau. Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving. In *Robot Learning*. J. Connell and S. Mahadevan (eds.). Kluwer Academic Publishing. 1993.

[42] D.A. Pomerleau. Input Reconstruction Reliability Estimation. In *Advances in Neural Information Processing Systems 5*. Giles, C.L., Hanson, S.J., and Cowan, J.D. (eds.). Morgan Kaufmann Publishers. San Mateo, CA. 1993.

[43] D.A. Pomerleau and D.S. Touretzky. Understanding Neural Network Internal Representations through Hidden Unit Sensitivity Analysis. *Proceedings of the International Conference on Intelligent Autonomous Systems*. C.E. Thorpe (ed.). IOS Publishers, Amsterdam. 1993.

[44] D.A. Pomerleau. Progress in Neural Network-based Vision for Autonomous Robot Driving. In *Proceedings of the 1992 Intelligent Vehicles Symposium*. I. Masaki (ed.). pp 391-396. 1992.

[45] D.A. Pomerleau, J. Gowdy, C.E. Thorpe. Combining artificial neural networks and symbolic processing for autonomous robot guidance. In *Proceedings of the 1992 DARPA Image Understanding Workshop*. 1992.

[46] D.A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. In *Neural Computation* 3(1). pp. 88-97. 1991.

[47] D.A. Pomerleau. Neural network-based vision processing for autonomous robot guidance. In *Proceedings of SPIE Conference on Aerospace Sensing*. Orlando, FL. 1991.

[48] D.A. Pomerleau. Rapidly Adapting Artificial Neural Networks for Autonomous Navigation. In *Advances in Neural Information Processing Systems 3*. R.P. Lippmann, J.E. Moody, and D.S. Touretzky

(ed.). Morgan Kaufmann, pp. 429-435. 1991.

[49] D.A. Pomerleau, J. Gowdy, and C.E. Thorpe. Combining artificial neural networks and symbolic processing for autonomous robot guidance. *Journal of Engineering Applications of Artificial Intelligence*. Chris Harris, (Ed.). 1991.

[50] D. Reece. *A Tactical Control System for Mobile Robot Driving*. PhD Dissertation, Carnegie Mellon. 1992.

[51] D. Reece and S. Shafer. *A Computational Model of Driving for Autonomous Vehicles*. CMU-CS-91-122. April 1991.

[52] L. Robert and M. Hebert. Deriving Orientation Cues from Stereo Images. *Proceedings European Conference on Computer Vision*. 1994.

[53] L. Robert and M. Hebert. Weakly-Calibrated Stereo Perception for Rover Navigation. *Proceedings International Conference on Computer Vision*. Cambridge, MA. 1995.

[54] J.K. Rosenblatt. DAMN: A Distributed Architecture for Mobile Navigation. In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. H. Hexmoor & D. Kortenkamp (Eds.). AAAI Press, Menlo Park, CA. 1995.

[55] J.K. Rosenblatt and C.E. Thorpe. Combining Multiple Goals in a Behavior-Based Architecture. *Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS)*. Pittsburgh, PA. 1995.

[56] B. Ross. A Practical Stereo Vision System. *Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR 93)*. New York, NY. 1993.

[57] A. Stentz. The Focussed D* Algorithm for Real-Time Replanning. *Proceedings of IJCAI-95*. 1995.

[58] A. Stentz and M. Hebert. A Complete Navigation System for Goal Acquisition in Unknown Environments. *Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS)*. Pittsburgh, PA. 1995.

[59] A. Stentz and M. Hebert. A Complete Navigation System for Goal Acquisition in Unknown Environments. *Autonomous Robots*. Vol. 2, No. 2. 1995.

[60] A. Stentz. Optimal and Efficient Path Planning for Unknown and Dynamic Environments. *International Journal of Robotics and Automation*. Vol. 10, No. 3. 1995.

[61] A. Stentz. *The D* Algorithm for Real-Time Planning of Optimal Traverses*. Carnegie Mellon University Robotics Institute Technical Report CMU-RI-TR-94-37, September, 1994.

[62] A. Stentz. Optimal and Efficient Path Planning for Partially-Known Environments. *Proceedings IEEE International Conference on Robotics and Automation*. 1994.

[63] A. Stentz, B.L. Brumitt, R. Coulter, A.J. Kelly. An Autonomous System for Cross-Country Navigation. *Proceedings of SPIE Symposium on Mobile Robots*. 1992.

[64] R. Sukthankar. *RACCOON: A Real-time Autonomous Car Chaser Operating Optimally at Night*. CMU-RI-TR-92-13, October 1992.

[65] R. Sukthankar, D.A. Pomerleau, C.E. Thorpe. *Panacea: An Active Sensor Controller for the ALVINN Autonomous Driving System*. CMU-RI-TR-93-09, April 1993.

[66] R. Sukthankar. RACCOON: A Real-time Autonomous Car Chaser Operating Optimally at Night. In *Proceedings of IEEE Intelligent Vehicle '93*. Tokyo. 1993.

[67] C. Thorpe, H. Athanassiou, J. Kay, T. Mitchell, and D. Pomerleau. Machine Learning and Human Interface for the CMU Navlab. In *Proceedings 6th Int. Symposium on Robotics Research*. T. Kanade and R. Paul, eds. 1994.

[68] C.E. Thorpe. Keynote Address: Point-Counterpoint: Big Robots VS Little Robots. *Proceedings 21st SPIE Applied Imagery Pattern Recognition Workshop*. Washington DC. 1992.

[69] C.E. Thorpe, O. Amidi, J. Gowdy, M. Hebert, D.A. Pomerleau. Integrating position measurement and image understanding for autonomous vehicle navigation. In *Proceedings of the Second International Workshop on High Precision Navigation*. Stuttgart, Germany. 1991.

[70] C.E. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Toward Autonomous Driving: The CMU Navlab. Part I: Perception. *IEEE Expert*. V 6 # 4. 1991.

[71] C.E. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Toward Autonomous Driving: The CMU Navlab. Part II: System and Architecture. *IEEE Expert*. V 6 # 4. 1991.

[72] C.E. Thorpe and J. Gowdy. Annotated Maps for Autonomous Land Vehicles. In *Proceedings of DARPA Image Understanding Workshop*. 1990.

[73] C.E. Thorpe and J. Crisman. UNSCARF, A Color Vision System for the Detection of Unstructured Roads. In *Proceedings of IEEE International Conference on Robotics and Automation*. 1991.

[74] C.E. Thorpe. Outdoor visual navigation for autonomous robots. In *Robotics and Autonomous Systems 7*. 1991.

Chapter II: Architecture and Tools

Introduction

Systems for unmanned vehicles are large collections of modules for perception, planning, and control. In this chapter, we describe our approach for building architectures that allow us to build complex systems from collections of modules.

The key to developing effective autonomous systems is an architecture that provides a framework within which individual components can be combined into a coherent system. One of the most useful means of classifying architectures is by their degree of centralization. At one end of the scale are behaviorist architectures, such as Brooks' Subsumption, which treat systems as loosely-connected collections of independent agents. Tasks such as obstacle avoidance or path following may each have their own sensors, their own set of real-time reactions, and their own output specifying which direction to follow. Each module is isolated from all other modules and there is no internal representation of the world. A fixed arbitration scheme decides which module is permitted to steer the vehicle at any one instant.

At the opposite extreme of the architecture spectrum are map-based architectures, which are built around a central module that contains the system's current model of the world. The system is decomposed into individual sensor modules, which add to the centralized world model, and into planning modules that use the world model to plan and execute actions. Since all modules work on the same representation, the planners and execution monitors are guaranteed to be consistent. Furthermore, since the best possible sensor data is all fused into one coherent model, the planners can select the optimal course of action.

In practice, neither of these approaches is necessarily the right one for building large systems. Subsumption systems are guaranteed to be fast, but may be difficult to coordinate for working on complex tasks. Since they do not have a central world model, it is hard to guarantee that individual modules are not working at cross-purposes. Also, their lack of symbolic world models makes it hard to communicate with subsumptionist systems in symbolic terms. Map-based systems often suffer from a performance bottleneck, since all information must flow through the map. Integrating data into the map can be slow, making it difficult to build fast-acting reflexes. It is also difficult to keep the map up to date and consistent. For example, one of our earlier map-based architectures was useful for prototyping big systems but was not easily extensible to fast-moving robots.

The CMU UGV architecture does not strictly follow either extreme of the architectural continuum. Instead, it embodies a pragmatic three-part philosophy:

- Model the behavior of a module: Some of our modules use neural nets, some geometric reasoning, some symbolic manipulation. What is important is not how the component works, but how its capabilities can be modeled so that other modules can use its output and provide its input.

- Build architectural toolkits: Some modules have internal maps; others have no state. Some modules run in milliseconds; others are slower, but can give partial results as needed. We prefer to build tools that let individual modules be integrated in a variety of styles, and that allow different systems to be built according to different principles.

- Expand behaviorist integration to include symbolic information: Modules should be able to cross levels of abstraction. We extract symbolic information from low-level modules, such as using a neural net to generate road descriptors; and we use symbolic planning to control the lower-level behaviors.

The first tool for developing a decentralized architecture is a communication library that allows efficient com-

munication between modules. A communication toolkit called IPT which is responsible for the communication in all of our systems. IPT is the most recent version of our communication software. Earlier versions of IPT, such as EDDIE and TCX, have also been used for building systems. Those earlier versions have similar functionality to that of IPT.

IPT consists of a server and a library. The server is used only for establishing communication links between modules. Any two modules may request a link by using IPT library functions. Once a link is established, the two modules may exchange information directly through asynchronous "receive" and "send" calls. The IPT library includes primitives for high-level description of message formats so that modules may exchange complex information.

IPT uses Unix sockets for the low-level communication. As a result, IPT can be ported to any Unix machine. It also handles communications through Ethernet as well as through shared memory. It is also optimized to handle communications between modules running on the same machines.

In addition to the basic receive/send mechanism, IPT includes a publish/subscribe mechanism in which a module can publish a particular type of message which will be automatically sent to any module that subscribes to that message. This mechanism is very important in hiding from a given modules the details of which module is using its information. For example, an obstacle detection module may publish obstacle maps which can be used transparently by any other module that subscribes to the appropriate message. The alternative is to force the obstacle detection module to establish explicit connections with all the modules that needs the obstacle information. Therefore, the publish/subscribe mechanism improves the flexibility of the architecture considerably. In order to properly handle multi-vehicle operation, we have modified IPT so that it can simultaneously manage several systems running on different vehicles. In that case modules may send messages to other modules on the same vehicle or they may send messages directly to another module on another vehicle.

In a typical system, several modules may have their own ideas of how the vehicle should be controlled. For example, a road follower wants to steer the vehicle according to its current idea of the position of the road in the image; an obstacle avoidance behavior tries to steer the vehicle away from the obstacle detected in the current range image. Therefore, we need a vocabulary that can be used by all the behaviors to describe their idea of the best steering and speed command. The basic vocabulary is a discrete set of steering arcs. Based on its current information, each behavior generates a vote for each of the arcs based. An arbiter receives the distributions of votes from all the modules, combines them and, based on the aggregate vote distribution, computes the best steering arc.

The arbiter and the library used by the behaviors for generating the vote distribution is the DAMN (Distributed Architecture for Mobile Navigation) system. DAMN is the basic foundation of all of our systems. It is used for controlling both the steering and the speed of the vehicle. In addition, we have augmented DAMN to handle sensor control. In that case, the vocabulary used by the behaviors is a discrete set of sensor orientations and the corresponding fields of view. A separate arbiter combines the votes for sensor orientations from different modules and controls the sensor orientation. We have included a detailed description of DAMN as the first section of this chapter.

Most recent additions to the arbiter include the ability to set limit on steering from any behavior for safety purposes and the use of a grid representation for the arc evaluation. The goal of the grid representation is to eliminate occasional oscillatory behavior of the arbiter and to provide a unified framework for arc evaluation.

The DAMN arbiter provides a simple and effective means of real-time control of autonomous driving. However, additional tools are needed in order to specify and monitor higher-level mission goals. Specifically, a high-level plan executor/monitor that reflects the realities of building an outdoor mobile navigation system was needed. To satisfy these needs, we built SAUSAGES (System for AUtonomous Specification, Acquisition, Generation, and Execution of Schemata.) In SAUSAGES, plans can be written for sequencing and controlling the execution of the behaviors used in a system. For example, SAUSAGES plans may include instructions such as "Change the parameters of behavior A when position X is reached", or "Activate behavior B whenever

behavior A has failed." At run-time, SAUSAGES controls all the behaviors by sending control messages of the appropriate type based on its plan.

SAUSAGES provides a flexible tool for mission planning, execution and monitoring. It has been used successfully for building our most complex systems with dozens of modules. We describe SAUSAGES in detail in the second part of this Chapter. The development of SAUSAGES was based in large part on an earlier system, the Annotated Maps system, in which states in a mission plan were specified entirely based on vehicle position. The Annotated Map system is also briefly described in the second part of the Chapter.

IPT, DAMN, and SAUSAGES together form the architectural glue used for building all of our systems from the behaviors described later in this document. This toolkit allows to build and configure systems without substantial modifications of the component modules by hiding the details of the behaviors through the use of a common vocabulary in DAMN. The toolkit allows the use of behaviors with disparate computational profiles through the use of point-to-point communications in IPT. Finally, it allows for control and monitoring of the behaviors at a symbolic level using SAUSAGES. This architectural toolkit satisfies the three goals listed above for a decentralized, behavior-based architecture.

A Behavior-Based Architecture for Mobile Navigation

1. Introduction

In order to function in unstructured, unknown, or dynamic environments, a mobile robot must be able to perceive its surroundings and generate actions that are appropriate for that environment and for the goals of the robotic system. Mobile robots need to combine information from several different sources. For example, the CMU Navlab vehicles are equipped with sensors such as video cameras, laser range finders, sonars, and inertial navigation systems, which are variously used by subsystems that follow roads and track paths, avoid obstacles and rough terrain, seek goals, and perform teleoperation. Because of the disparate nature of the raw sensor data and internal representations used by these subsystems, combining them into one coherent system which includes all their capabilities has proven to be very difficult.

To function effectively, an architectural framework for these sensing and reasoning processes must be imposed to provide a structure with which the system may be developed, tested, debugged, and understood. However, because the architecture should serve as an aid rather than a burden in the integration of modules that have been developed independently, it must not be overly restrictive. It must allow for purposeful goal-oriented behavior, yet retain the ability to respond to potentially dangerous situations in real time while maintaining enough speed to be useful.

Deliberative planning and reactive control are equally important for mobile robot navigation; when used appropriately, each complements the other and compensates for the other's deficiencies. Reactive components provide the basic capabilities which enable the robot to achieve low-level tasks without injury to itself or its environment; deliberative components provide the ability to achieve higher level goals and to avoid mistakes which could lead to inefficiencies or even mission failure.

Hierarchical approaches allow slower abstract reasoning at the higher levels and faster numerical computations at the lower levels, thus allowing varying trade-offs between responsiveness and optimality as appropriate at each level [11] [1]. While such an approach provides aspects of both deliberative planning and reactive control, the top-down nature of hierarchical structures tends to overly restrict the lower levels [13]. In hierarchical architectures, each layer controls the layer beneath it and assumes that its commands will be executed as expected; this introduces a need to monitor the progress of desired actions and to report failures as they occur [20], thus increasing complexity.

Rather than imposing a top-down structure to achieve this desired symbiosis of deliberative and reactive elements, the Distributed Architecture for Mobile Navigation takes an approach where multiple modules concurrently share control of the robot by sending votes to be combined rather than commands to be selected [16].

2. The Distributed Architecture for Mobile Navigation

Figure 1 shows the organization of the Distributed Architecture for Mobile Navigation (DAMN), in which individual behaviors such as road-following or obstacle-avoidance send votes to the command arbitration module; these inputs are combined and the resulting command is sent to the vehicle controller. Each action-producing module, or *behavior*, is responsible for a particular aspect of vehicle control or for achieving some particular task; it operates asynchronously and in parallel with other behaviors, sending its outputs to the arbiter at the appropriate rate for that particular function. Each behavior is assigned a weight reflecting its relative priority in controlling the vehicle. Based on knowledge of which behaviors would be most relevant and reliable in a given situation, a mode manager may also be used to vary these weights during the course of a mission.

Like other distributed architectures, DAMN enjoys several advantages over a centralized one, including greater reactivity, flexibility, and robustness. While all votes must pass through the command arbiter before an action

is taken, the function provided by the arbiter is fairly simple and does not represent the centralized bottleneck of more traditional systems. However, unlike reactive systems such as the Subsumption Architecture [3], the perception and planning components of a behavior are not prohibited from maintaining complex internal representations of the world. It has been argued that "the world is its own best model" [4], but this assumes that the vehicle's sensors and the algorithms which process them are essentially free of harmful noise and that they can not benefit from evidence combination between consecutive scenes. In addition, disallowing the use of internal representations requires that all environmental features of immediate interest be visible to the vehicle sensors at all times, unnecessarily constraining and reducing the flexibility of the overall system.

2.1. Command Arbitration

In a distributed architecture, it is necessary to decide which behaviors should be controlling the vehicle at any given time. In some architectures, this is achieved by having priorities assigned to each behavior; of all the behaviors issuing commands, the one with the highest priority is in control and the rest are ignored [3] [18]. In order to allow multiple considerations to affect vehicle actions concurrently, DAMN instead uses a scheme where each behavior votes for or against each of a set of possible vehicle actions [17]. An arbiter then performs *command fusion* to select the most appropriate action. While the Motor Schema framework [2] also offers a means of fusing commands from multiple behaviors, it suffers from the well known problem of local minima in potential fields. Another, perhaps more serious problem, is that arbitration via vector addition can result in a command which is not satisfactory to any of the contributing behaviors. DAMN arbiters do not average commands, but rather select the command which has the most votes from the behaviors.

Each behavior generates a vote between -1 and +1 for each possible action; in the case of the turn arbiter, the possible actions are a discrete set of steering commands. At each iteration, the arbiter collects any new votes that the behaviors may have sent since the last cycle, computes a normalized weighted sum of the votes, and determines which command has the maximum vote value. In order to avoid problems with discretization such as biasing and "bang-bang" control (i.e., alternating between discrete values in order to achieve an intermediate value), the arbiter performs sub-pixel interpolation. This is done by performing Gaussian smoothing, selecting the command option with the highest value, and then fitting a parabola to that value and the ones on either side; the peak of that parabola is used as the command to be issued to the controller. This process, which is repeated at a rate of 10 Hz, is illustrated in Figure 2, where the votes from two behaviors (a & b) are linearly combined (c), and then smoothed and interpolated to produce the resulting command (d). This is similar to defuzzification in Fuzzy Logic systems [9]; indeed an architecture has been implemented which recasts DAMN into a Fuzzy Logic framework [24].

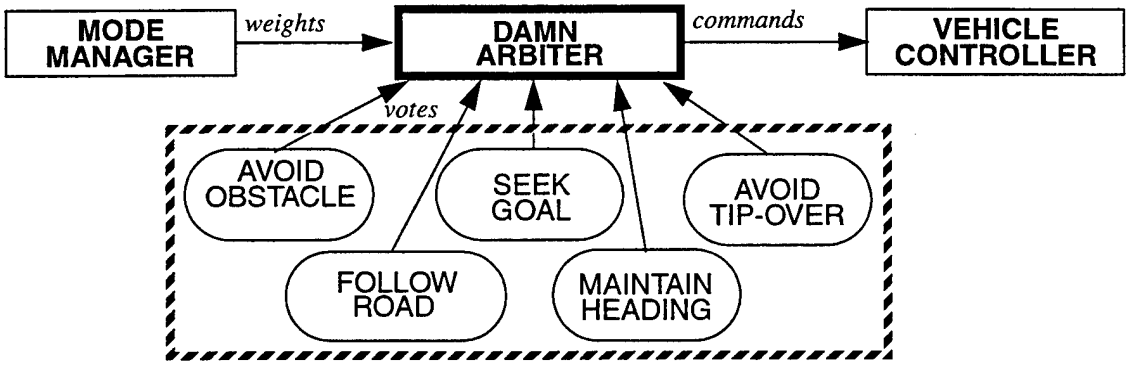


Figure 1: Behaviors sending votes to arbiter.

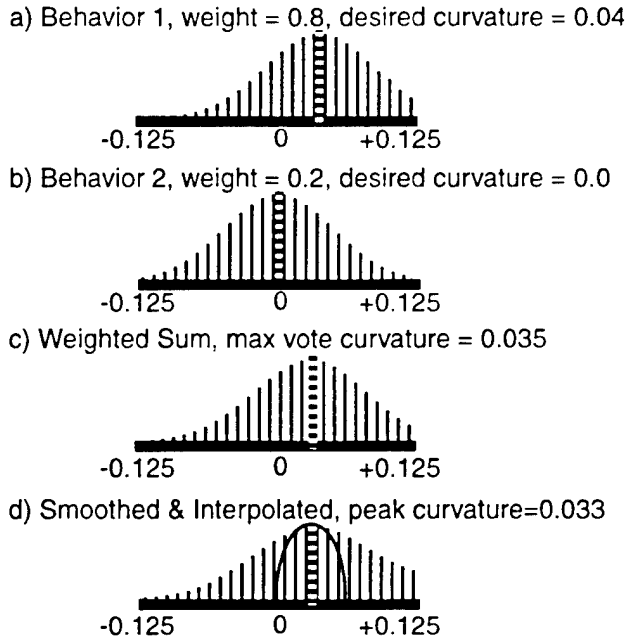


Figure 2: Command fusion process.

A field-of-regard arbiter and its associated behaviors have also been recently implemented and used for the control of a pair of stereo cameras on a pan/tilt platform. Behaviors vote for different possible field-of-regard polygons (the camera field of view mapped on to the ground plane); the arbiter maintains a local map of these votes and transforms them as the vehicle moves. At each iteration, these votes are mapped into a pan-tilt space, and then smoothed and interpolated as described above. Figure 3 shows the votes for field of regard polygons generated by a behavior that votes against looking in the direction of known obstacles since travelling in that direction is impossible. Behaviors also vote based on considerations such as looking toward the goal and looking at a region contiguous to already mapped areas. The darkest polygon in the figure corresponds to the pan and tilt angles selected by the arbiter.

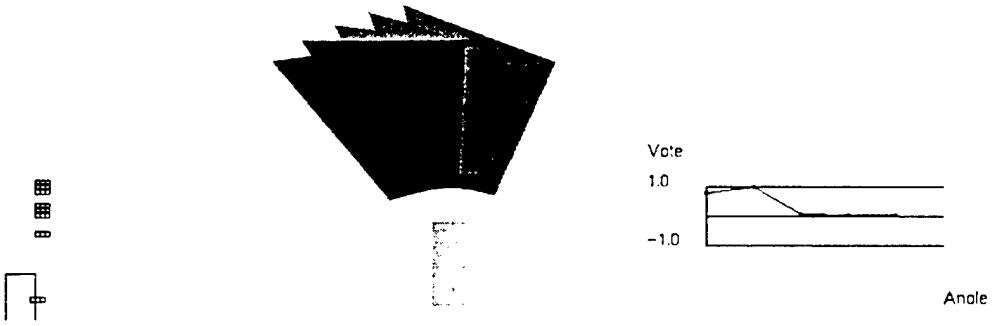


Figure 3: Field of regard voting and arbitration; (a) Local map around the vehicle; (b) Field-of-regard polygons shaded according to the votes assigned by the field of regard behavior; (c) Votes for panning angles.

2.2. Behaviors

Within the framework of DAMN, behaviors must be defined to provide the task-specific knowledge for controlling the vehicle. Since behaviors may be arbitrarily deliberative or reflexive, DAMN is designed so that behaviors can issue votes at any rate; for example, a reflexive behavior may operate at 10 Hz, another behavior may maintain local state and operate at 1 Hz, while a global planner may issue votes at a rate of 0.1 Hz. The use of distributed shared control allows multiple levels of planning to be used in decision-making without the need for an hierarchical structure. However, higher-level reasoning modules may still exert meta-level control within DAMN by modifying the voting weights assigned to behaviors.

Safety Behaviors. A basic need for any mobile robot system is the ability to avoid situations hazardous to itself or to other objects in its environment. Therefore, an important part of DAMN is its "first level of competence"[3], which consists of behaviors designed to provide vehicle safety. In contrast to priority-based architectures which only allow one behavior to be effective at any given moment, the structure of DAMN and its arbitration scheme allow the function of these safety behaviors to be preserved as additional levels of competence are added.
The *Obstacle Avoidance* behavior receives a list of current obstacles in vehicle-centered coordinates and evaluates each of the possible command options, as illustrated in Figure 4. The source of these obstacles may be intraversable regions of terrain determined by range image processing or stereo vision, by sonar detection of objects above the ground plane, or any other means of obstacle detection as appropriate to the current task and environment [5][8].

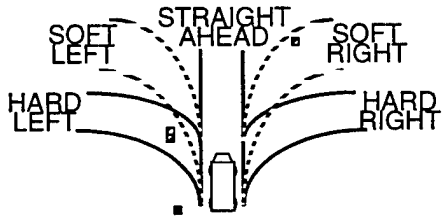


Figure 4: Arc evaluation in the *Obstacle Avoidance* behavior

Another vital aspect of vehicle safety is insuring that the commanded speed and turn stay within the dynamic constraints of the vehicle as it travels over varying terrain conditions. The *Limit Turn* behavior sends votes to the arbiter that reflect these constraints, voting against commands that violate them.

Road Following. Once vehicle safety has been assured by the obstacle avoidance and dynamic constraint behaviors, it is desirable to integrate task-level modules such as the ALVINN road following system [15]. In the case of ALVINN, creating a behavior that independently evaluated each arc was relatively straightforward. The output units of a neural network represent evaluations of particular turn commands; these units are simply resampled to the DAMN turn vote space, using a Gaussian of the appropriate width. This process is illustrated in Figure 5.

Teleoperation. The STRIPE teleoperation system [6] allows an operator to designate waypoints for the vehicle; STRIPE determines an appropriate steering commands and sends a set of votes representing a Gaussian centered on the desired command. This allows the dynamic constraints and obstacle avoidance behaviors to be used in conjunction with STRIPE so that the safety of the vehicle is still assured.

Goal-Directed Behaviors. While the lower-level behaviors operate at a high rate to ensure safety and provide functions such as road following and obstacle avoidance, higher-level behaviors are free to process information at a slower rate and periodically issue votes to the arbiter that guide the robot towards the current goal.
The *Goal Seeking* behavior is one way to provide this capability. This simple behavior directs the vehicle toward a series of goal points specified in global coordinates either by the user [8] or by a map-based planner

[7]. The desired turn radius is transformed into a series of votes by applying a Gaussian whose peak is at the desired turn radius and which tapers off as the difference between this turn radius and a prospective turn command increases.

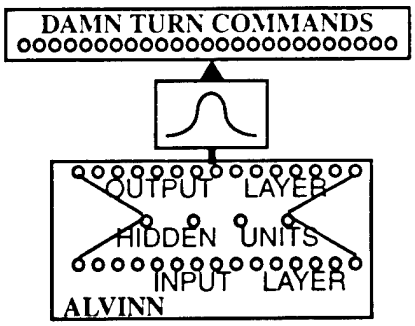


Figure 5: Resampling of ALVINN output layer.

Some more sophisticated map-based planning techniques based on the A* search algorithm [10] have also been integrated and used within the DAMN framework. These planners use dynamic programming techniques to determine an optimal global path; however, an important point is that they do not hand a plan down to a lower level planner for execution, but rather maintain an internal representation that allows them to participate directly in the control of the vehicle based on its current state.

For example, the D* planner [21] creates a grid with information on how best to reach the goal from any location in the map. The global plan is integrated into DAMN as a behavior by determining, for each possible turn command, the distance to the goal from a point along that arc a fixed distance ahead (as shown in Figure 6).

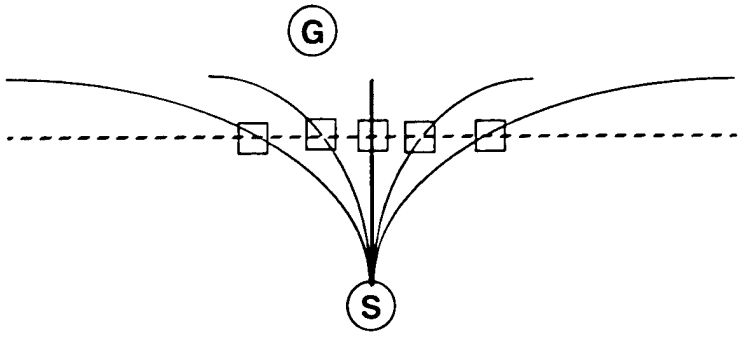


Figure 6: Using D* to evaluate distance from goal.

2.3. Evolutionary Development and Integration

DAMN is designed so that various behaviors can be easily added or removed from the system, depending on the current task at hand. Although the modules described above all use very different paradigms and representations, it has been relatively straightforward to integrate each and every one of them into the framework of DAMN. Sensor fusion is not necessary since the command fusion process in DAMN preserves the information that is critical to decision-making, yielding the capability to concurrently satisfy multiple objectives without the need for centralized bottlenecks. A detailed description of an implemented system and the experimental results achieved can be found in [8].

The voting strengths, or weights, of each behavior are specified by the user or by a mission-planning module. DAMN is fairly insensitive to the values of these weights and the system performs well without a need to tweak these parameters. For example, the *Obstacle Avoidance* behavior and the *Seek Goal* behavior have been used with weights of 0.75 and 0.25 and of 0.9 and 0.1, respectively, and in both cases the robot successfully reached its goal while avoiding obstacles. The voting weights of each behavior can also be modified by a mode manager module such as the Annotated Maps system, which was integrated with DAMN to provide this capability [23].

All of the behaviors described in Section 2.2. have been used in conjunction with each other in various configurations, yielding systems that were more capable than they would have been otherwise. Conceptually, three levels of competence have been implemented in DAMN thus far, as shown in Figure 7. These levels of competence are convenient for describing the incremental manner in which the system's capabilities evolve; however, it is important to note that all behaviors co-exist at the same level of planning. The importance of a behavior's decisions is reflected by the weighting factor for its votes, and is in no way affected by the level of competence in which it is described.

The safety behaviors are used as a first level of competence upon which other levels can be added. Movement is the second level of competence that has been implemented; road following, cross-country navigation, and teleoperation behaviors have all been run together with the obstacle avoidance behavior to provide various forms of generating purposeful movement while maintaining safety [23]. The third level of competence is comprised of the various map-based goal-seeking behaviors. Cross-country behaviors have been combined with goal-oriented behaviors to produce directed off-road navigation [7] [13] [21].

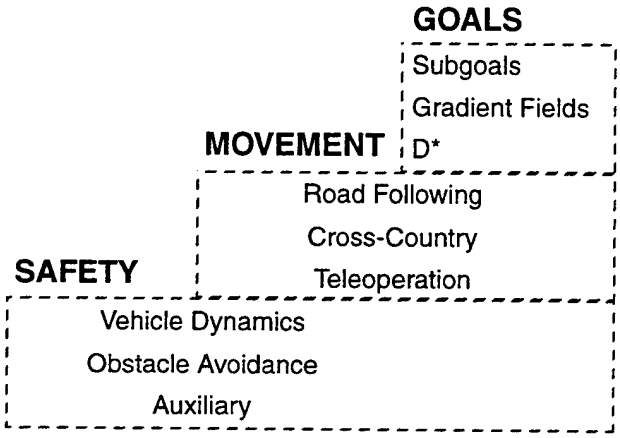


Figure 7: Evolutionary development in DAMN.

2.4. Future Work

DAMN has proven to be very effective as an architecture which greatly facilitates the integration of a wide variety of different vehicle navigation subsystems; however, as DAMN is used for increasingly complex tasks where many behaviors may be issuing votes concurrently, there will be a greater need to have the semantics of the voting process defined more carefully. By explicitly representing and reasoning about uncertainty within the decision-making processes, a system can be created whose effects are well-defined and well-behaved.

Utility theory is a means of defining the semantics of the voting and arbitration processes. Because we are attempting to decide which among a set of possible actions to take, it is natural to make judgments on the usefulness of each action based on its consequences. If we assign a utility measure $U(c)$ for each possible conse-

quence of an action, then the *expected utility* for an action a is:

$$U(a) = \sum_{c} U(c) \cdot P(c|a, e)$$

where *P(cla,e)* is the probability distribution of the consequence configuration c, conditioned upon selecting action a and observing evidence e [14]. Thus, if we can define these utilities and probabilities, we can then apply the *Maximum Expected Utility* (MEU) criterion to select the optimal action based on our current information.

If utility theory is to be applied to the problem of evidence combination and action selection for mobile navigation tasks, a means for defining the utility functions must be provided. Behaviors are defined in order to achieve some task, so it is fair to assume that there must be at least an implicit measure of "goodness" or utility with respect to that task. For example, an obstacle avoidance behavior's task is to maximize distance to obstacles, so that the distance from the vehicle to the nearest obstacle could be used as a measure of goodness. Likewise, proximity to the current goal could be used for the goal-based behaviors, as proximity to the center of a road (or lane) could be used by road following modules.

Future research should also include command fusion using a local map for turn and speed arbitration, as was done for the field of regard arbiter. This will allow the arbiter to ensure consistency in the voting process by transforming votes as the vehicle moves. This will also allow behaviors to vote on where to travel rather than how to steer; however, complications due to the non-holonomic constraints of the vehicle will need to be addressed.

Another issue that needs to be addressed is the coupling between turn and speed commands. Currently, the turn and speed arbiters operate independently. Turn behaviors can use vehicle speed as one of their inputs and vice versa; however, there is currently no mechanism which allows behaviors to vote for a combination of turn and speed. Research is ongoing to determine a command representation which allows both coupled and independent voting, and is still efficient enough for real-time purposes.

3. Conclusion

The Distributed Architecture for Mobile Navigation has been successfully used to create systems which safely follow roads or traverse cross-country terrain while avoiding obstacles and pursuing mission goals. DAMN imposes no constraints on the nature of the information or the processing within a behavior, only on the interface between the behavior and the command arbiter. Furthermore, the behaviors are not subject to timing constraints; each behavior operates asynchronously and in parallel. In addition to its use on the CMU Navlab vehicles, DAMN has also been used at Martin Marietta, Hughes Research Labs, and Georgia Institute of Technology.

Like centralized or hierarchical architectures, DAMN is able to assimilate and use high-level information. Non-reactive behaviors may use plans to achieve goals and coordinate with other agents. However, like other behavior-based architectures, it also avoids sensory and decision-making bottlenecks and is therefore able to respond in real-time to external and internal events. Finally, unlike architectures with strictly prioritized modules, DAMN's vote arbitration scheme allows multiple goals and constraints to be fulfilled simultaneously, thus providing goal-oriented behavior without sacrificing real-time reactiveness.

Bibliography

[1] J. Albus, H. McCain, R. Lumia. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*. NBS Tech. Note 1235. Gaithersburg, MD. 1987.

[2] R. Arkin. Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior. *Proceedings International Conference on Robotics and Automation.* 1987.

[3] R. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation.* vol. RA-2, no. 1. 1986.

[4] R. Brooks. Intelligence Without Reason. *Proceedings IJCAI.* 1993.

[5] M. Daily, J. Harris, D. Keirsey, K. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong. Autonomous Cross-Country Navigation with the ALV. In *Proceedings IEEE Conference on Robotics and Automation.* Philadelphia, PA. 1988.

[6] J.S. Kay, C.E. Thorpe. STRIPE Supervised Telerobotics Using Incremental Polygonal Earth Geometry. In *Proceedings Intelligent Autonomous Systems Conference.* 1993.

[7] D.M. Keirsey, D. Payton, and J.K. Rosenblatt. Autonomous Navigation in Cross-Country Terrain. In *Proceedings of Image Understanding Workshop.* Cambridge, MA. 1988.

[8] D. Langer, J.K. Rosenblatt, and M. Hebert. A Behavior-Based System For Off-Road Navigation. In *IEEE Journal of Robotics and Automation.* Vol. 10, No. 6. 1994.

[9] C. Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller -- Parts I & II. *IEEE Transactions on Systems, Man and Cybernetics.* Vol 20 No 2. 1990.

[10] N. Nilsson. *Shakey the Robot.* SRI Tech. Note 323. Menlo Park, CA. 1984.

[11] D.W. Payton. An Architecture for Reflexive Autonomous Vehicle Control. In *Proceedings IEEE International Conference on Robotics and Automation.* San Francisco, CA. 1986.

[12] D.W. Payton. Internalized Plans: A Representation for Action Resources. *Robotics and Autonomous Systems.* 6(1). 1990.

[13] D.W. Payton, J.K. Rosenblatt, D.M. Keirsey. Plan Guided Reaction. *IEEE Transactions on Systems Man and Cybernetics.* 20(6). 1990.

[14] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers. 1988.

[15] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Ph.D. dissertation. Carnegie-Mellon Technical Report CMU-CS-92-115. 1992.

[16] J.K. Rosenblatt. DAMN: A Distributed Architecture for Mobile Navigation. In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents.* H. Hexmoor & D. Kortenkamp (Eds.). AAAI Press. Menlo Park, CA. 1995.

[17] J.K. Rosenblatt and D.W. Payton. A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks.* Washington DC. 1989.

[18] S.J. Rosenschein and L.P. Kaelbling. The Synthesis of Digital Machines with Provable Epistemic Properties. In *Proceedings Theoretical Aspects of Reasoning about Knowledge.* 1986.

[19] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press. 1976.

[20] R. Simmons, L.J. Lin, C. Fedor. Autonomous Task Control for Mobile Robots. *Proceedings IEEE Symposium on Intelligent Control.* Philadelphia, PA. 1990.

[21] A. Stentz. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments.* Carnegie-Mellon Technical Report CMU-RI-TR-93-20. 1993.

[22] A. Stentz, M. Hebert. *A Complete Navigation System for Goal Acquisition in Unknown Environments.* Carnegie-Mellon Technical Report CMU-RI-TR-94-7. 1994.

[23] C. Thorpe, O. Amidi, J. Gowdy, M. Hebert, D. Pomerleau. Integrating Position Measurement and Image Understanding for Autonomous Vehicle Navigation. *Proceedings Workshop on High Precision Navigation.* Springer-Verlag Publisher. 1991.

[24] J. Yen, N. Pfluger. A Fuzzy Logic Based Robot Navigation System. *Proceedings AAAI Fall Symp.* 1992.

SAUSAGES: Between Planning and Action

1. Introduction

Several years ago, it became apparent that some form of mission execution and monitoring was needed to integrate the capabilities of the perception systems. We had road followers that robustly followed roads [6], object detectors that avoided obstacles [3], landmark recognizers that localized the vehicle position in a map[3], but we had no consistent architectural glue to join them. A robust road follower is impressive on its own, but a road follower alone has no way to know which way to turn at an intersection, no way to know when to speed up or slow down for important events, etc. A system that can execute a complex mission cannot simply be the sum of its perceptual modalities; there needs to be a "plan" which uses high-level knowledge about goals and intentions to direct the behaviors of the low level perception and actuation modules.

The driving force behind the development of the task mission execution and monitoring software is the challenge of getting a real system to work on a real robot. We have to work within the realities of the perception subsystems that exist. The system that resulted from these constraints is SAUSAGES: the System for AUtonomous Specification Acquisition, Generation, and Execution of Schemata.

2. The road to SAUSAGES

An intuitive way to represent a plan and execute it is through events, e.g., "Follow this road until the second intersection, turn left, and stop at a mail box." This plan representation works well for simulated and indoor mobile robots, which operate in a highly structured and controlled environment where every event necessary to the plan execution is easily perceived. Unfortunately, once a robot system leaves the laboratory it is in a much less structured, less controlled, and less predictable environment. The events that an outdoor mobile robot system needs to detect to execute a plan are very difficult to discriminate from the background environment. Reliable and accurate acquisition of perceptual cues, such as intersections or signs, is an almost impossible task. A system that relies solely on these perceptual events to direct its plan is not feasible.

To get around this problem and still have a system that could execute interesting missions, Annotated Maps[11] approach was developed. An Annotated Map system reduces the problem of monitoring for arbitrary events, such as intersections or signs, to the problem of monitoring the vehicle position on an a priori map. The plan in an Annotated Map is essentially a positional production system, with productions like "at position x, perform action y" instead of productions like "if event x then perform action y." This alternate representation of plans vastly reduces the computational resources necessary to perform a mission, while still allowing complex scenarios.

For example, we used this system to drive the Carnegie Mellon Navlab from one house in a suburb north of Pittsburgh to another approximately one kilometer away. The system used neural networks to drive the vehicle along the roads and used a laser range finder to detect obstacles and landmarks. The system successfully performed the mission, speeding up and slowing down at the appropriate places, navigating through several intersections, and coming to a stop at the proper mailbox.

Figure 1 shows the annotated map we used in this system. We built this map by driving the vehicle over the course, storing the road position received from the controller and the landmarks detected by the laser range finder. A human expert then went through and added "annotations," or trigger lines. When the vehicle crosses these lines various actions are taken, such as setting speed, or turning modalities on or off. Thus, these trigger lines implicitly specify the plan.The process of navigating through an intersection shows how this simple geometric implicit plan specification works. We had no perception system that allowed us to navigate an intersection, so it had to be done using only the map, and thus the vehicle's position had to be accurate. Figure 2 gives an example of this process. As the vehicle's position is tracked going down the road, the first trigger line that it

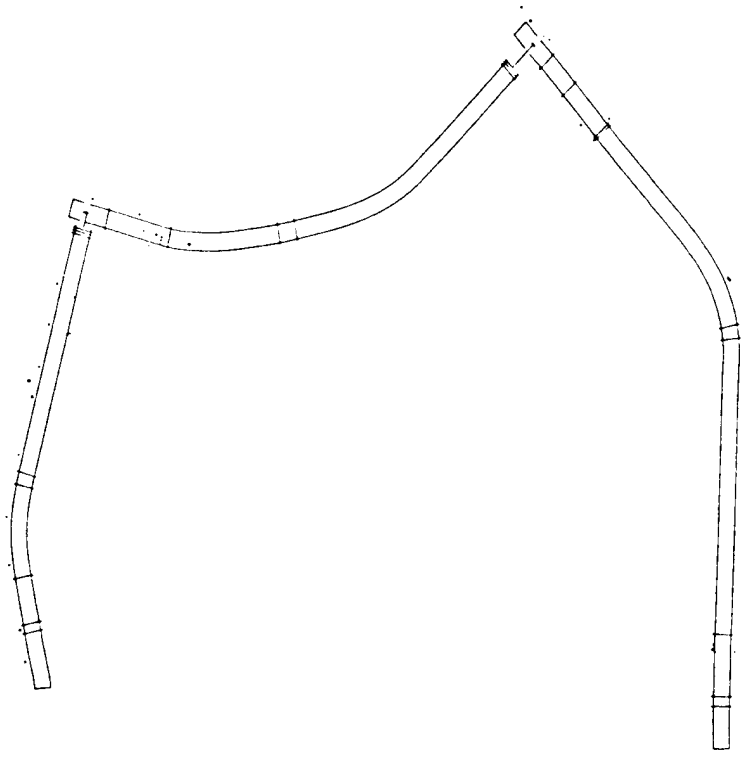


Figure 1: A sample annotated map.

crosses will turn on landmark-based position updating to pinpoint the vehicle's position on the map for precise navigation. Then the vehicle will cross a series of lines that will slow it down in stages. Once the vehicle actually enters an intersection, it crosses a line which turns off road following and turns on dead reckoning from the map to actually make a left turn. Once the vehicle leaves the intersection, it crosses lines that turn road following back on, turn landmark-based position updating off, and sets the vehicle speed back to maximum again. Each step in this "plan" is sequenced implicitly by the positioning of the trigger lines and assumptions about the motion of the vehicle

There are several possible objections to using a global coordinate system with an absolute map. What if the map is wrong? What if the positioning system drifts? We have found that these are not valid objections if there

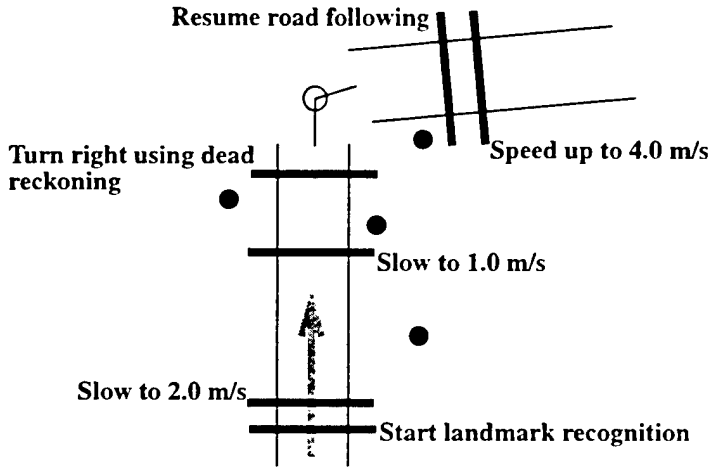


Figure 2: Navigating an intersection with an Annotated Map.

are landmark recognition systems that can guarantee that the map is locally correct during the execution of the mission. It does not matter if the map is wrong globally as long as the region currently around the vehicle is roughly correct. What is important is that the control information encoded by the trigger lines is tied geometrically to actual features in the map, such as roads or intersections. If the position of the vehicle relative to a local landmark is accurate, then the position of the vehicle relative to the control information will be accurate, and therefore the plan will be accurately executed.

Having these landmark recognition systems is not much of an additional expenditure of resources since the perception systems needed to navigate the vehicle can double as sources of information about landmarks. For example, obstacle avoiders can double as discrete landmark detectors. Even road followers can be used to pinpoint the vehicle's position in a map with surprising ease and precision. As Figure 3 shows, a road follower can reduce the uncertainty in the vehicle's position perpendicular to the axis of the road. If the vehicle rounds a corner, the cumulative updates in the vehicle's position on the map will reduce the uncertainty of the vehicle's position dramatically[12].

The underlying philosophy in building outdoor mobile robot systems is that perception is the primary task, i.e., the task that the most computational and developmental resources will be expended on. Planning has been considered secondary, since it is a much "easier" problem. Annotated maps are an example of this philosophy taken to an extreme. Annotated Maps allow to demonstrate our perception and actuation modules in complex scenarios with a minimum of resource investment in the event detection and planning issues.

This approach is in direct contrast to much of the previous work in the field in that many researchers consider the interesting part of their system to be the planning, monitoring, and error recovery capabilities. The perception systems are an afterthought to showcase the abilities of the planning system. This can be seen in the plethora of indoor mobile robots and simulations found in the literature in which the environment can be structured and controlled to the advantage of the plan generation, execution, and monitoring systems.

Either extreme has obvious limitations. A system that reasons very robustly and intelligently about events that it cannot robustly detect in the real world will be an interesting system with no real application. At the other extreme, a system that only uses Annotated Maps for mission planning will rigidly perform one pre-planned mission, but won't react well to unexpected changes in the plan, goals, or environment.

For example, while an Annotated Maps system can handle repeated, well established scenarios, such as mail delivery or a daily commute, it cannot begin to execute a simple, non-geometric plan for the first time, such as, "go to the first corner and turn left." For every mission there has to be an a priori, detailed map and a priori

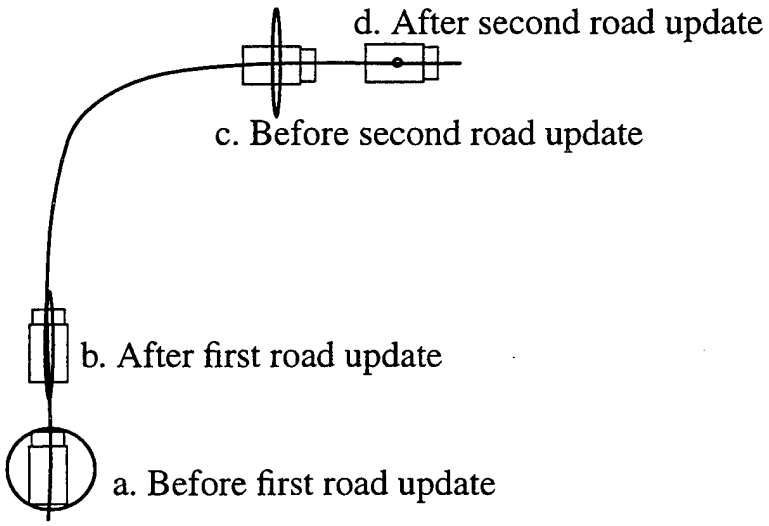


Figure 3: Using a road follower to update positions on a map.

expert knowledge of how to navigate around in that map. Another major limitation of an Annotated Maps "plan" is that it is only implicit, i.e., the sequencing of the "steps" of the plan are implicit to the geometric placement of the annotations. The Annotated Map manager itself has no concept of "plan", just of the map and the vehicle position. This makes it difficult to execute missions with loops in the plan without bending, if not breaking, the paradigm. It also makes it hard to monitor for failures of the plan. How is the map manager supposed to detect a failure in the plan when it doesn't even explicitly know what the plan is?

Current experience building real outdoor systems shows that we need a high level plan executor/monitor to integrate our various perception and actuation modules into coherent systems. Also, any such high level knowledge system needs to reflects the realities of building an outdoor mobile navigation system: What the system can do is driven primarily by what it can perceive, not what it can plan. At the same time any plan executor/monitor has to be much more flexible and reactive than Annotated Maps. To satisfy these needs we built SAUSAGES.

3. Description of SAUSAGES

SAUSAGES stands for System for AUtonomous Specification, Acquisition, Generation, and Execution of Schemata. SAUSAGES is designed to be a bridge between the worlds of planning and perception. It can be used to integrate perception modules together into useful systems with a minimum of planning work, and it is easy to use with external planners that can generate, monitor, and adjust SAUSAGES plans to create intelligent and reactive behavior in the face of changing environments and goals.

A SAUSAGES plan is made of discrete semantic units that we call "links." While in an Annotated Map, the sequencing of the plan is implicit, while in a SAUSAGES plan the sequencing is explicit. Where an annotated map plan is a geometric map with annotations, a SAUSAGES plan is a directed graph of links.

A link can be thought of as a single step in a plan. A link has entrance actions that get invoked when the link starts up, and exit actions to clean up when the link exits. The link has a set of production rules associated with it that are added to a global production rule system. These production rules have actions that specify when the link is finished, when it has failed, side affects of the link, or whatever the system designer wants them to specify. These production rules are only in effect while the link is active.

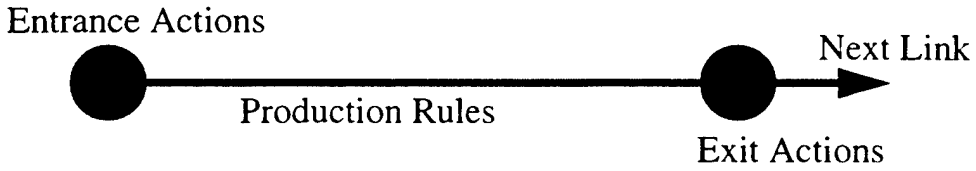


Figure 4: A link of SAUSAGES.

SAUSAGES uses a Link Manager process that manages the execution of links. It is the Link Manager that maintains the production rule system and starts and stops links. Most interaction with a SAUSAGES plan is done through the Link Manager by setting state variables that fire various production rules. The Link Manager also lets external modules remotely add links, run them, delete them, and even change them.

A SAUSAGES link also has a function that returns the next link to execute when it is finished. This is how a planner hooks together a series of links into a plan. This function can be as simple as returning another link, which might be part of a serial chain of links, or it could be conditional, where if one condition holds activate one link and if another holds activate another. This allows a plan writer to construct plans that are directed graphs of links rather than just a serial chain. Thus, a planner can create a plan that has some intelligence and decision making built into it to handle common contingencies that might be foreseen.

Links can be composed of sub-links to form hierarchical plans. This is a common abstraction in plan descrip-

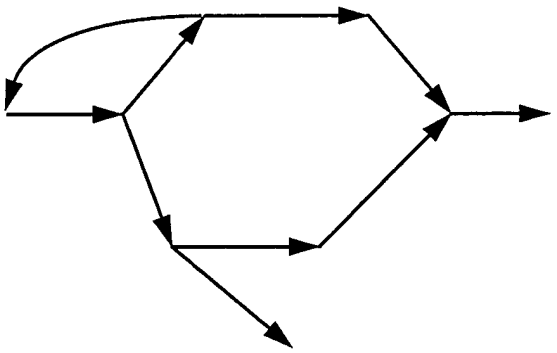


Figure 5: A plan as a graph of links.

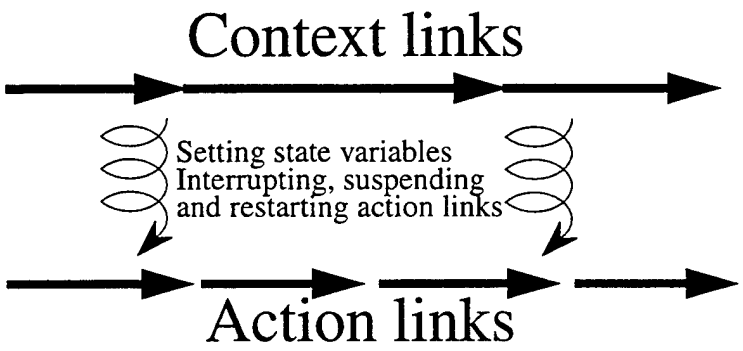


Figure 6: Concurrent links.

tions. A link which implements "go to airport," will decompose into several sub-links, such as "get in car," "drive to airport," and "park at airport." Each of these links could have their own production rules with completion and failure conditions attached. The plan designer can specify that any failure be handled entirely at the level of the link that failed, or that failures get propagated up the link hierarchy.

SAUSAGES plans do not have to be "plans" in the traditional sense. They can simply be algorithms. An external planner can produce an algorithm for a situation, complete with looping or recursion, download it to SAUSAGES, and run it. We get this flexibility because the "next link" function is arbitrary, and instead of producing a predetermined link, it could be a function that generates a new link based on the current link or any condition that is relevant. Thus, SAUSAGES "algorithms" can be recursive and dynamic rather than simply iterative and static.

The plan graph can have more than one link active at a time. This way the Link Manager can simulate multiple threads of control. One common example for mobile robots is to have two threads of control, an action thread and a context thread. The action thread will control the robot motion and perception, while the context thread will monitor for global errors or major changes in the plan. The context thread can affect how the action thread transitions. The context thread can start and stop links in the action thread, or replace links in the action thread entirely.

So, SAUSAGES provides a planning language that is well suited for specifying and executing mobile robot plans. These plans can range in complexity from a simple linear linking together of different steps in a plan to actual algorithms including looping and recursion. A great advantage of the SAUSAGES approach is that plans are specified in semantically meaningful units which can be easily manipulated by external planners and monitors in useful ways.

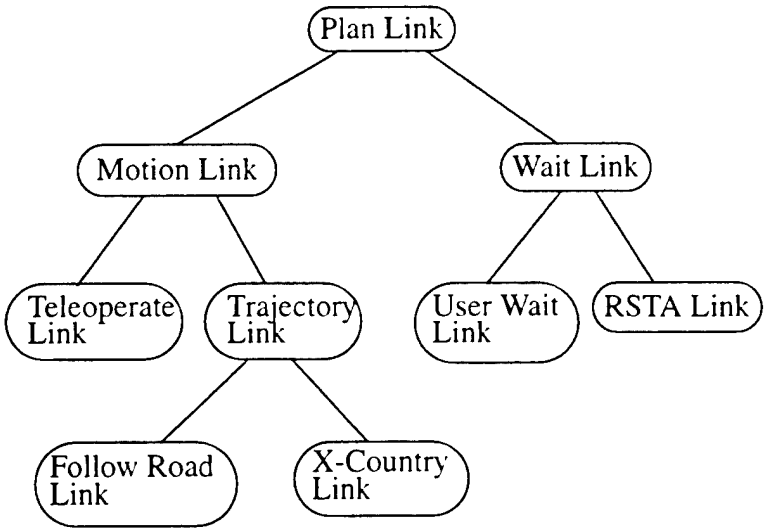


Figure 7: The UGV link class hierarchy.

4. Implementation of SAUSAGES

Although SAUSAGES is written in C++, SAUSAGES plans, links, and interface functions are specified using a C++ Lisp interpreter. The most notable variation of this Lisp is that it has a simple class system which can be especially useful in specifying types of "links" for the Link Manager. For example, there can be a "trajectory" link class that follows a trajectory and contains much of the administrative work for following trajectories. Then subclasses of the trajectory class can be specified that follow roads or go cross country. At present, we only allow inheritance of data and methods from a single parent class, but Figure 7 shows how even single inheritance can create a useful class hierarchy.

Another key feature of our interpreted Lisp is that it is trivial to write new C++ functions and to invoke them from Lisp expressions. This extensibility supports the philosophy of using Lisp as an interface and doing any computationally expensive work in C++. This takes advantage of the run-time strengths of both languages: Lisp's flexibility and C++'s efficiency.

Using Lisp as the interface to SAUSAGES allows great flexibility in specifying a plan or algorithm. Link entrance and exit actions are naturally specified by Lisp expressions. Arbitrarily complex Lisp expression's can be used in specifying "rule" criteria. Classes can be used as link templates, so a "follow-road" link is just an instance of the follow road link class with the appropriate data slots filled in and the remaining data slots filled with default values.

The most important module in SAUSAGES is the Link Manager, which actually manages the execution and monitoring of links. Link networks, i.e. plans, that the Link Manager will run can be read in from files or transmitted from other modules via a TCP/IP connection.

In order to run links and affect their behavior, the Link Manager maintains a simplified production rule system. Each rule in the system has a list of tags, a criteria expression (the "left hand side" of the rule), and an action expression to evaluate when the criteria expression evaluates to non-nil. A rule's criteria expression is only evaluated when one of its tags is marked as changed. This is not a complete forward propagating rule system, but it has proven sufficient for the purposes of mobile robot planning. If a complete production system is needed it will be easy to add and integrate. The Link Manager also maintains a global timer table which maintains a list of expressions to be evaluated at regular intervals

These global systems for productions and timers are the means by which a SAUSAGES plan can seem to be multi-threaded. When a link is run, the Link Manager adds the link's rules and timers to the global systems and when the link finishes they are removed. If two links are active "concurrently", both links' rules and timers are active in the global systems.

External modules can easily send the Link Manager expressions to be evaluated, and thus can easily affect the firing of rules in the global production system. The remote expression evaluation provides one method for an external module to create and run links remotely. The Link Manager also provides methods for exporting references to the links themselves so that external modules can modify them as needed. The Link Manager can also send messages to external modules just as easily as it gets them. These are just some of the ways a SAUSAGES system can interact with external perception modules and plan monitors.

The only kind of events monitored in an Annotated Maps systems were based on the position of a vehicle on a map. We could write these map monitoring capabilities directly into the Link Manager, but we find it more convenient to delegate this processing to a separate module called the Navigator, since this allows us to experiment with different position maintenance and updating approaches. The Navigator has two jobs:

1. Maintaining a transform between the position that the vehicle controller reports and the position on a map.

2. Handling client queries about the vehicle's position relative to objects in the world. Clients can register a geometric query and the Navigator will inform the client when the vehicle has moved to a position that triggers that query. This query mechanism can be used for such purposes as monitoring the crossing of trigger lines and checking to make sure the vehicle stays within a tolerance distance of a trajectory.

5. SAUSAGES and Annotated Maps

It is obvious in SAUSAGES how to represent event-based plans: Each link terminates on the occurrence of some event, such as "intersection detected," or "Fred's mailbox found." Unfortunately, this brings up the original problem with this type of plan representation for outdoor mobile robots: Detecting the arbitrary events necessary to proceed from plan step to plan step. We can address this problem by using segments of Annotated Maps as individual links.

In the simplest possible annotated map link, the only production rule would be, "If vehicle X crosses the line

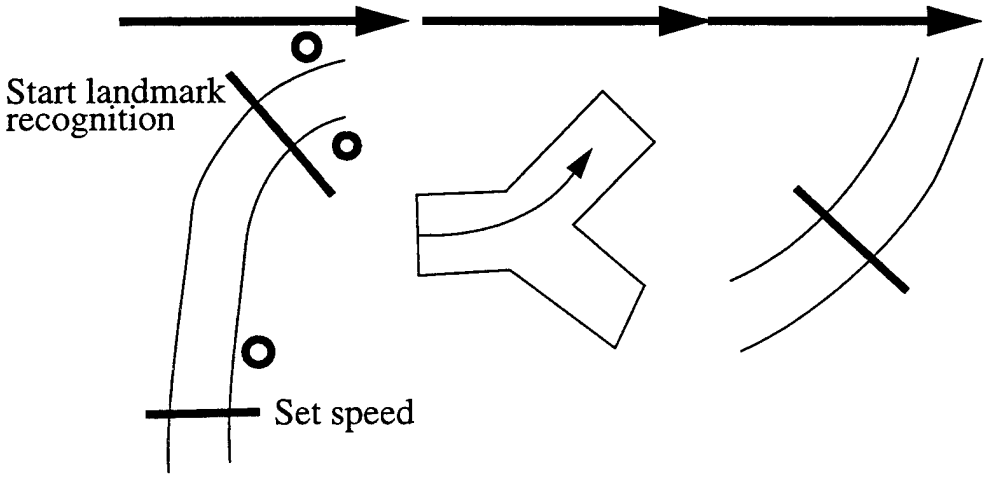


Figure 8: Using Annotated Maps as SAUSAGES links.

(x1, y1), (x2, y2), end the link and go to the next one." The next link would just be the next segment of the Annotated Map. Stringing links of this type together exactly simulates the capabilities of our current Annotated Map system, with the plan made more explicit.

Using SAUSAGES to make an Annotated Maps plan explicit frees a system from many of the limitations of Annotated Maps while retaining the central advantage of Annotated Maps: the reduction of the event detection problem to a position monitoring problem. For example, retrotraversing a trajectory was difficult with Annotated Maps since the system had to discriminate between the annotations relevant to the journey out and the annotations relevant to the journey back. SAUSAGES easily solves this problem by allowing two annotated map links with the same geometric information about the world, but with different annotation data for each leg of the traversal. So each link has the same road and landmark data but with different control information.

Since each step of the plan is an explicit link, the user can attach explicit and different failure conditions to each step. Thus we can easily and flexibly specify the conditions under which a plan fails and what to do when the plan fails. For example, in a road following annotated map link we would want a failure condition to be "if the vehicle position is too different from the road position, fail." For a cross-country annotated map link, we would want the same condition, but with a higher tolerance. This flexibility in specifying the failure conditions is trivial to implement in a SAUSAGES plan because each class of link can have a different set of production rules that indicate errors. With Annotated Maps alone, implementing even this simple error monitoring requires adding code that does not fit the design well.

All links are equivalent to SAUSAGES, so it is trivial to mix "geometric" or map based links with "cognitive" or event- based links. A SAUSAGES plan is not locked into either paradigm and provides extensive support for both. This allows planners to easily generate plans that can both retraverse known territory using map-based links and explore new territory using event-based links, assuming that event detectors can be built.

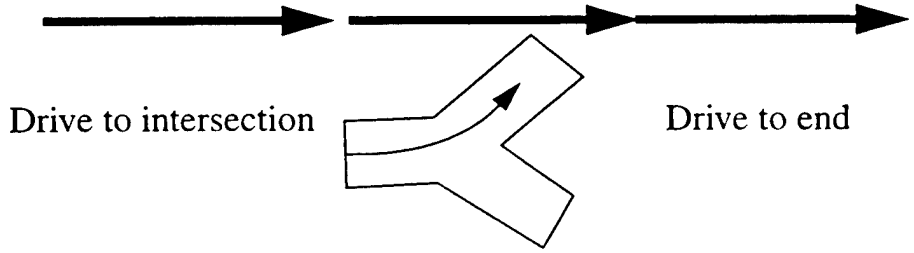


Figure 9: Mixing geometric and cognitive links.

6. SAUSAGES and planners

SAUSAGES is not a planning system. It is a plan executor and monitor that stands between a planner and the real world, between the world of symbols and propositions and the world of pixels and actuators. It depends on external sources for its plans, whether that be a human hand coding a mission plan for a specific demo or a sophisticated AI planner generating plans as it responds to changes in environment and goals.

One of the historical models of planning involves a separation of planning and execution[5]. A planner comes up with a complete plan, sends it to an executor which does its best to perform the plan, and reports success or failure to the planner. In this paradigm there is a conceptual wall between planning and execution, and many

researchers have shown the limitations of this approach. Is SAUSAGES a step backwards to this paradigm?

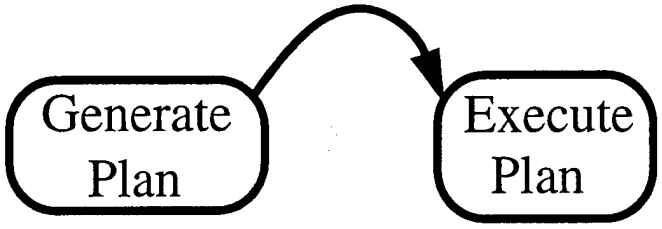


Figure 10: Separation of planning and execution.

It is easy to construct SAUSAGES systems using this paradigm, but SAUSAGES is not limited to it. SAUSAGES was designed for heavy interaction with any number of external planners, not just one omniscient planner. External modules can create, edit, and run links. Links can be designed to report status back to external modules and can have data associated with them that is not necessary for plan execution, but which do encode information necessary to manipulate the plan.

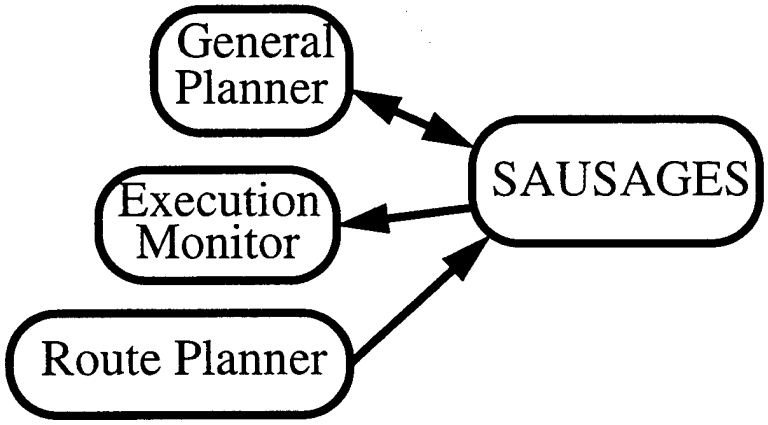


Figure 11: SAUSAGES interacting with external planners and monitors.

A SAUSAGES plan does not have to be a passive, static graph, since the "next link" function can create new links at run time. The relevant case for planning is a next link function that generates a call to an external planner asking what to do now. The most extreme form of this approach would be to make every link's next function report status back to an external planner and request that the planner generate the next link.

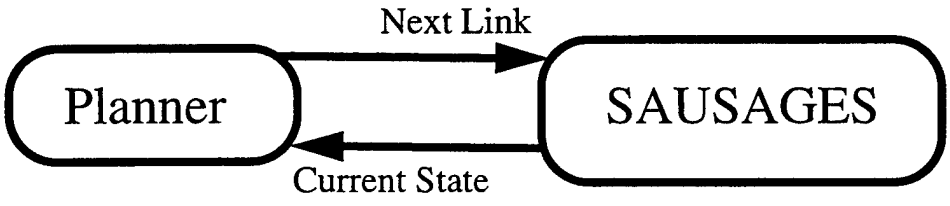


Figure 12: A SAUSAGES system that completely interleaves planning and execution.

This approach, which totally interleaves planning and execution, can be used to interface a SAUSAGES system with incremental planners such as the Procedural Reasoning System (PRS)[2] which may use the universal planner approach [7]. At each step in a plan a incremental planner examines everything it knows about the situation before applying the next action. This planning paradigm lets the system easily respond to errors, sys-

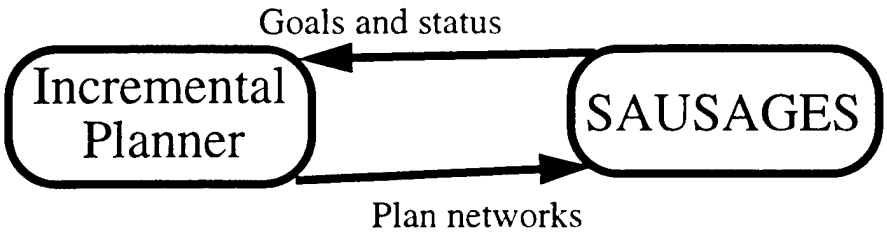


Figure 13: The ultimate SAUSAGES intelligent agent.

tem failures, goal changes, and other unexpected changes in the world. It also lets the "plan" be mostly incomplete and underspecified. Using this planning/execution paradigm in a SAUSAGES system essentially reduces the role of SAUSAGES to a "mode manager," i.e., the planner spits out a high level command such as "follow the road using system X" which SAUSAGES translates into actual commands to perception and actuation modules. SAUSAGES is also used in such a system to translate failures of perception and actuation modules into symbology that the incremental planner can process.

While this approach is robust, it is not the proper approach to use in a SAUSAGES system. It throws away many of the abilities of SAUSAGES in order to have everything controlled by a single monolithic planner. We find in building systems that putting too much of an emphasis on general purpose modules results in degraded real-time performance. It is much better to have multiple specialized systems, with each system doing its job efficiently and robustly. SAUSAGES is not a general purpose module: its specialization is as a plan executor and monitor, and any computationally expensive job should be delegated to an external module. The "incremental plan" scenario does not take advantage of the things that SAUSAGES does well.

A better approach for the ultimate SAUSAGES system is to still use something very much like PRS, but instead of generating single plan steps the system would generate algorithms and plan sequences that are appropriate for achieving the goals of the mission. This plan graph sent to SAUSAGES would incorporate the necessary information to monitor and to detect failure. Any failures would be handled by reporting to the Planner the new state of the world so that it could generate a new algorithm or plan sequence to achieve the goals of the system with the new knowledge of the failure.

This approach takes advantage of the "reactive" abilities of an incremental planner like PRS while still exploiting the efficiency and flexibility of SAUSAGES. The question of which system is dominant, the incremental planner or the SAUSAGES plan executor, is irrelevant. It doesn't really matter whether there is a SAUSAGES plan network that invokes the incremental planner which in turn generates SAUSAGES programs, or if there is an incremental planner which generates a SAUSAGES program which calls out to the planner, which in turn generates more SAUSAGES programs.

A human problem solver executing a task does not typically think about each step. A human will see a goal to achieve and will try and find a schema or algorithm that achieves that goal. Typically, a human problem solver will only realize that an algorithm is not appropriate when it fails, and only then will the slow, thoughtful step by step "planning" to achieve the goal or work around it be tried. This behavior, known as Einstellung effect [4], corresponds well to a system that uses an incremental planner to choose SAUSAGES plan networks. The incremental planner has a goal, and to achieve that goal it generates a SAUSAGES network. The SAUSAGES network runs until it finishes or fails. If the network successfully finishes, the planner can go on to working on the next goal. If the network fails, the planner can then go into a more intensive step by step incremental planning mode, or it can take whatever other course of action it deems necessary.

7. Conclusions

SAUSAGES' potential has been tested on unmanned ground vehicles both at Carnegie Mellon and Martin

Marietta. Systems using SAUSAGES for task management have driven ground vehicles on missions of several kilometers using a mix of road following, obstacle avoidance, and remote teleoperation.

Our experience shows that SAUSAGES allows incremental development of real outdoor mobile robot systems. Groups working primarily in perception and actuation can use powerful constructs such as annotated maps to quickly generate systems that perform interesting missions with a minimum of work in high-level planners. Groups working primarily on user interfaces and high-level planning can use SAUSAGES to interact with a real robot on the symbolic level rather than the pixel and motor level.

It is easy to see how SAUSAGES allows the development of a complete outdoor mobile robotic system that is driven by what the robot can perceive rather than what the planners want to perceive. As the perception and actuation modules improve, the developers create SAUSAGES links and algorithms to perform various tasks. When a planning system matures to the point of being integrated with a real robot, it can use these various SAUSAGES links and algorithms that have been heavily tested in the field as building blocks for plans.

Another advantage of SAUSAGES is that it does not lock the system designer into any single planning or architectural paradigm. It is simply a bridge between planning and action, and does not specify how either is to be accomplished, or if either is to happen at all. SAUSAGES lets the system designer vary the level and extent of involvement of a general planner in the execution of a plan.

For example, suppose that one leg of a mission has a pair of vehicles performing a "bounding overwatch." A bounding overwatch is a maneuver in which two vehicles mutually support each other as they travel under the possible observation of an "enemy." A bounding overwatch cannot be completely specified as a set of trajectories and mode transition points, since the exact execution depends on a large number of environment variables and constraints that can not be determined a priori. A SAUSAGES system could implement a bounding overwatch by having a general purpose reactive planner keep track of the current goals and states of the vehicles and enemy. This planner would generate new links one by one as the bounding overwatch progresses. Alternatively, if a system designer can design a bounding overwatch algorithm it would be more efficient to have a specialized bounding overwatch module implement this algorithm. In this case, the higher level planner just generates a link, "do a bounding overwatch," which SAUSAGES interprets as "ask the bounding overwatch planner to do a bounding overwatch." If the algorithm for doing a bounding overwatch is simple enough, the bounding overwatch module could be completely dispensed with, and the bounding overwatch could be done through an algorithm implemented as a SAUSAGES network. With SAUSAGES, any of these could easily be implemented, and it is up to the system designer to decide which is best.

The philosophy in building SAUSAGES systems is to avoid generalization. Instead of having one monolithic general planner that handles all situations completely, it is better to set up a planning system that has a general purpose planner that generates links that call domain specific planners that are much more efficient at planning in their domain. There is still a place for a general high level planner, but it should not be involved at all levels of decisions. It is a much better approach to use the most efficient representation and solution methods to achieve a goal, and then have a well-defined way of combining the results of these methods. In our systems, SAUSAGES is that well-defined way.

This philosophy comes from a desire to avoid bottlenecks in knowledge and communications. Simmons claims that having an omniscient central module that controls everything and that all data must flow through makes monitoring and control easier [8], but it reflects unrealistic assumptions about bandwidth and real time response to environmental changes. SAUSAGES appears to reside in the center of a system, between planners and perception modules, but it does not suffer from the bottleneck problem. In a properly designed SAUSAGES system, SAUSAGES selects which modules are active and gives them advice. These modules then go about the business of keeping the mobile robot alive and do not depend on real-time responses from SAUSAGES.

For example, SAUSAGES will be giving commands like, "start road following," and "try to follow this trajectory." The primary job of the road followers and cross-country navigators is to keep the robot safe, the second-

ary is to follow the advice. It is up to a SAUSAGES system to monitor the progress of the modules and to give them better advice if the advice turns out to be bad. The last things these modules should depend on for real-time response is the SAUSAGES planner and its ancillary high-level planners. This approach is similar to Agre and Chapman's idea of plans as communication rather than programs: A SAUSAGES plan is a communication from the high level planners and user interfaces to the perception and actuation subsystems for the intended actions of the vehicle [1]. The actual actions of the vehicle result from an interaction between the plan, the environment, and the perceptual subsystems abilities.

Systems can be designed in which SAUSAGES is a bottleneck, but that never has to happen. For example, if there is a route planner that needs feedback from the terrain, and thus needs high-resolution terrain maps from a perception system, this information should not be transmitted through SAUSAGES. The route planner could use SAUSAGES to set up a direct connection between the planner and the perception module, and then use that connection to communicate this high-bandwidth information. SAUSAGES is designed to act as a system traffic cop, and not the information highway itself.

SAUSAGES provides a module that can be the middle ground between the extremes of the robotics world. It allows a researcher working on a high-level planner to avoid the awful implementation details of a real mobile robot system by abstracting the operation of the lower-level perception and actuation modules. It allows a perception researcher to easily build a system that showcases the perception work in a complex scenario without worrying too much about issues of planning and error recovery that are irrelevant to the task at hand. Finally, SAUSAGES facilitates the joining of these two extremes into a complete, intelligent, and robust outdoor mobile robot system.

Bibliography

[1] P. E. Agre and D. Chapman. What are plans for?. *IEEE Transactions on Robotics and Automation.* 617-34, 1990.

[2] M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings IEE Special Issue on Knowledge Representation*, pages 1383-1398. 1986.

[3] M. Hebert. Building and navigation maps of road scenes using an active sensor. *Proceedings International Conference on Robotics and Automation.* IEEE Computer Society. 1989.

[4] A. S. Luchins. Mechanization in problem solving. *Psychological Monographs.* 54(248). 1942.

[5] N. Nilsson. *Shakey the Robot.* Technical Report Tech. Note 323. SRI, Menlo Park, CA. 1984.

[6] D. Pomerleau. ALVINN: An Autonomous Land Vehicle In a Neural Network. *Advances in Neural Information Processing Systems I.* D. Touretzky (Ed.). Morgan Kaufmann. 1989.

[7] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. *Proceedings International Joint Conference on Artificial Intelligence.* 1987.

[8] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation.* 10(1):34-43. 1994.

[9] T. Stentz. *The NAVLAB System for Mobile Robot Navigation.* Ph.D. Thesis. Carnegie-Mellon University. 1989.

[10] C. Thorpe. *Vision and Navigation: the The Carnegie Mellon Navlab.* Kluwer Academic Publishers, 1990.

[11] C. Thorpe and J. Gowdy. Annotated Maps for Autonomous Land Vehicles. *Proceedings of DARPA Image Understanding Workshop.* Pittsburgh PA. 1990.

[12] C. Thorpe, O. Amidi, J. Gowdy, M. Hebert, D. Pomerleau. Integrating position measurement and image understanding for autonomous vehicle navigation. *Proceedings Second International Workshop on High Precision Navigation.* 1991.

Chapter III: Road Following

Introduction

The first set of mobility behaviors that we developed addresses the problem of autonomous road following. The problem is to control speed and steering using a single video camera so that the vehicle remains on the road. In the context of the UGV program, the two main requirements are:

- *Road types:* The road-following behavior must be able to operate on very different types of roads, from well marked highways to dirt roads. It must be able to handle large road curvature and intersections when necessary.

- *Speed:* Road following must be fast enough to drive the vehicle at normal speeds (e.g., 55 m.p.h. on highways.) In addition, we wanted the road-following behavior, like all the other behavior, to run on conventional computing hardware. This requirements puts severe constraints on the type of algorithms used for road following.

The most successful road follower is ALVINN (Autonomous Land Vehicle In a Neural Network). ALVINN is based on a neural network which takes an image of the road as input and produces a steering direction as output. ALVINN is trained by observing a human driver and by comparing the road images with the driver's steering commands.

ALVINN does not make any assumption on the appearance of the road or on its salient features. Therefore, it can handle a wide variety of roads. In fact, ALVINN has been demonstrated on multi-lane highways, urban streets, bicycle paths, and dirt roads.

Once the network is trained, ALVINN requires very little computation and can run on off-the-shelf workstations. As a result, ALVINN can achieve autonomous road following at highway speeds.

ALVINN, as it was originally developed, is described in detail in the first part of this Chapter. We built many new capabilities on top of the basic ALVINN in order to handle more complex situations and to improve its robustness. Among them:

- *Input reconstruction:* The IRRE (Input Reconstruction Reliability Estimation) metric measures how well the ALVINN network performs on a given input image. The IRRE provides a measure of confidence that can be used in order to decide whether the output of ALVINN should be ignored, for example, because the road is not in the image, and, in general, to evaluate the performance of ALVINN.

- *Multiple nets:* ALVINN can drive only on the road for which the neural network is trained. One extension of ALVINN is MANIAC (Multiple ALVINN Networks In Autonomous Control) in which the output of several ALVINN nets trained for different road types are combined through additional layers of units in order to compute the output steering command. MANIAC retains the computational simplicity of ALVINN and its basic performance yet it enables the use of the road follower on different types of road without modification or retraining. MANIAC was demonstrated by driving on several different road types (single-, two-, and four-lane roads) and showing that MANIAC was able to transition smoothly from one type of road to another.

- *Camera control:* One limitation of the original ALVINN is that the road may not remain visible in the image if the vehicle is steering too sharply. This happens, for example, on curved dirt roads. We have implemented a module for changing the orientation of the camera based on the current arc executed by the vehicle and on ALVINN output. This module, Panacea, was demonstrated on curved roads, showing its ability to keep the road centered in the camera image as the vehicle drives through the curves.

- *Virtual cameras:* Given proper calibration data, the image used by ALVINN can be transformed to a new image as would be seen from a different position and orientation of the camera, i.e., from a "virtual camera." Virtual cameras allow ALVINN to make predictions on the location of the road, to find branches of intersections, and to locate a road segment from an off-road position.

In addition to having been used successfully as part of the UGV integrated demos at Lockheed Martin, ALVINN has been used for convoying applications and other applications. More importantly, ALVINN and its descendants formed the basis for the Automated Highway Systems (AHS) program funded by DOT.

The last two parts of this chapter describe extensions to ALVINN. The first part is a detailed description of the core ALVINN system. The second part is a description of our most recent addition to ALVINN, the use of virtual cameras. The other additions to ALVINN are described in detail in the references.

Neural Network Vision for Robot Driving

1. Introduction

Autonomous navigation is a difficult problem for traditional vision and robotic techniques, primarily because of the noise and variability associated with real world scenes. Autonomous navigation systems based on traditional image processing and pattern recognition techniques often perform well under certain conditions but have problems with others. Part of the difficulty stems from the fact that the processing performed by these systems remains fixed across various environments.

Artificial neural networks have displayed promising performance and flexibility in other domains characterized by high degrees of noise and variability, such as handwritten character recognition [17], speech recognition [1] and face recognition [4]. ALVINN (Autonomous Land Vehicle In a Neural Network) is a system that brings the flexibility of connectionist learning techniques to the task of autonomous robot navigation.

This chapter describes the architecture, training and performance of the ALVINN system. It demonstrates how simple connectionist networks, when trained appropriately, can learn to precisely guide a mobile robot in a wide variety of situations. In particular, this chapter presents training techniques that allow ALVINN to learn in under 5 minutes to autonomously control the Navlab by watching a human driver's response to new situations. Using these techniques, ALVINN has been trained to drive in a variety of circumstances, including single-lane paved and unpaved roads, multilane lined and unlined roads, and obstacle-ridden on- and off-road environments, at speeds of up to 55 miles per hour.

2. Network Architecture

The basic network architecture employed in the ALVINN system is a single hidden layer feedforward neural network (See Figure 1). The input layer now consists of a single 30x32 unit "retina" onto which a sensor image from either a video camera or a scanning laser rangefinder is projected. Each of the 960 input units is fully connected to the hidden layer of 4 units, which is in turn fully connected to the output layer. The 30 unit output layer is a linear representation of the currently appropriate steering direction which may serve to keep the vehicle on the road or to prevent it from colliding with nearby obstacles. The centermost output unit represents the "travel straight ahead" condition, while units to the left and right of center represent successively sharper left and right turns. The units on the extreme left and right of the output vector represent turns with a 20m radius to the left and right respectively, and the units in between represent turns which decrease linearly in their curvature down to the "straight ahead" middle unit in the output vector.

To drive the Navlab, an image from the appropriate sensor is reduced to 30x32 pixels and projected onto the input layer. After propagating activation through the network, the output layer's activation profile is translated into a vehicle steering command. The steering direction dictated by the network is taken to be the center of mass of the "hill" of activation surrounding the output unit with the highest activation level. Using the center of mass of activation instead of the most active output unit when determining the direction to steer permits finer steering corrections, thus improving ALVINN's driving accuracy.

3. Network Training

The network is trained to produce the correct steering direction using the backpropagation learning algorithm [7]. In backpropagation, the network is first presented with an input, and activation is propagated forward through the network to determine the network's response. The network's response is then compared with the known correct response. If the network's actual response does not match the correct response, the weights between connections in the network are modified slightly to produce a response more closely matching the correct response.

Autonomous driving has the potential to be an ideal domain for a supervised learning algorithm, like back-

propagation, since there is a readily available teaching signal or "correct response" in the form of the human driver's current steering direction. In theory, it should be possible to teach a network to imitate a person as they drive using the current sensor image as input and the person's current steering direction as the desired output. This idea of training "on-the-fly" is depicted in Figure 2.

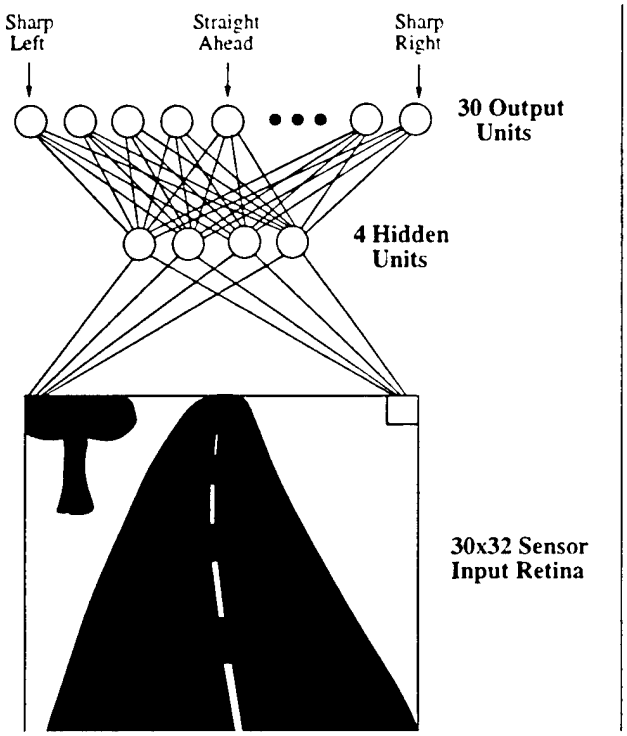


Figure 1: Neural network architecture for autonomous driving.

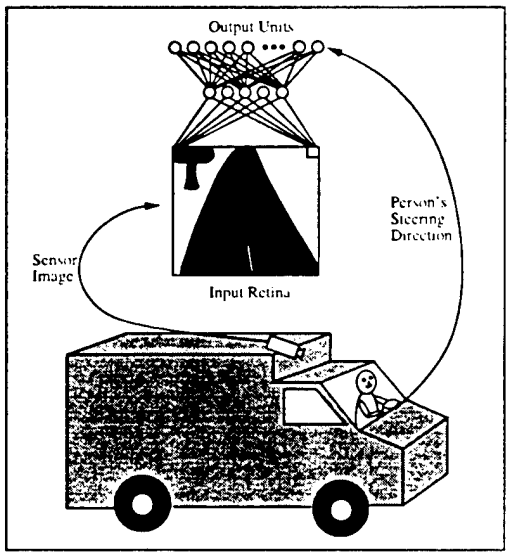


Figure 2: Schematic representation of training "on-the-fly". The network is shown images from the onboard sensor and trained to steer in the same direction as the human driver.

Training on real images would dramatically reduce the human effort required to develop networks for new sit-

uations by eliminating the need for a hand-programmed training example generator. On-the-fly training should also allow the system to adapt quickly to new situations.

3.1. Potential Problems

There are two potential problems associated with training a network using live sensor images as a person drives. First, since the person steers the vehicle down the center of the road during training, the network will never be presented with situations where it must recover from misalignment errors. When driving for itself, the network may occasionally stray from the road center, so it must be prepared to recover by steering the vehicle back to the middle of the road. The second problem is that naively training the network with only the current video image and steering direction may cause it to overlearn recent inputs. If the person drives the Navlab down a stretch of straight road at the end of training, the network will be presented with a long sequence of similar images. This sustained lack of diversity in the training set will cause the network to "forget" what it had learned about driving on curved roads and instead learn to always steer straight ahead.

Both problems associated with training on-the-fly stem from the fact that back-propagation requires training data which is representative of the full task to be learned. The first approach we considered for increasing the training set diversity was to have the driver swerve the vehicle during training. The idea was to teach the network how to recover from mistakes by showing it examples of the person steering the vehicle back to the road center. However this approach was deemed impractical for two reasons. First, training while the driver swerves would require turning learning off while the driver steers the vehicle off the road, and then back on when he swerves back to the road center. Without this ability to toggle the state of learning, the network would incorrectly learn to imitate the person swerving off the road as well as back on. While possible, turning learning on and off would require substantial manual input during the training process, which we wanted to avoid. The second problem with training by swerving is that it would require swerving in many circumstances to enable the network to learn a general representation. This would be time consuming, and also dangerous when training in traffic.

3.2. Solution - Transform the Sensor Image

To achieve sufficient diversity of real sensor images in the training set, without the problems associated with training by swerving, a technique for transforming sensor images to create additional training exemplars was developed. Instead of presenting the network with only the current sensor image and steering direction, each sensor image is shifted and rotated in software to create additional images in which the vehicle appears to be situated differently relative to the environment (See Figure 3). The sensor's position and orientation relative to the ground plane are known, so precise transformations can be achieved using perspective geometry.

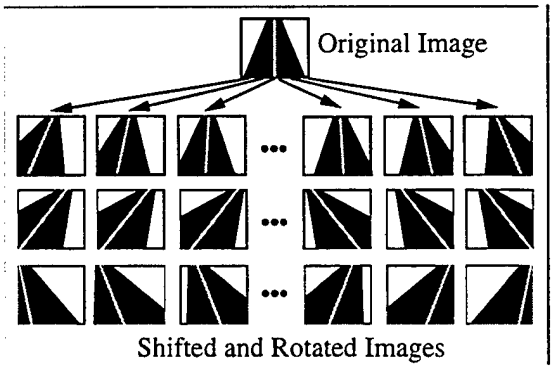


Figure 3: The single original video image is shifted and rotated to create multiple training exemplars in which the vehicle appears to be at different locations relative to the road.

The image transformation is performed by first determining the area of the ground plane which is visible in the original image, and the area that should be visible in the transformed image. These areas form two overlapping trapezoids as illustrated by the aerial view in Figure 4. To determine the appropriate value for a pixel in the transformed image, that pixel is projected onto the ground plane, and then back-projected into the original image. The value of the corresponding pixel in the original image is used as the value for the pixel in the transformed image. One important thing to realize is that the pixel-to-pixel mapping that implements a particular transformation is constant. In other words, assuming a planar world, the pixels that need to be sampled in the original image in order to achieve a specific shift and translation in the transformed image always remain the same. In the actual implementation of the image transformation technique, ALVINN takes advantage of this fact by precomputing the pixels that need to be sampled in order to perform the desired shifts and translations. As a result, transforming the original image to change the apparent position of the vehicle simply involves changing the pixel sampling pattern during the image reduction phase of preprocessing. Therefore, creating a transformed low resolution image takes no more time than is required to reduce the image resolution to that required by the ALVINN network. Obviously the environment is not always flat. But the elevation changes due to hills or dips in the road are small enough so as not to significantly violate the planar world assumption.

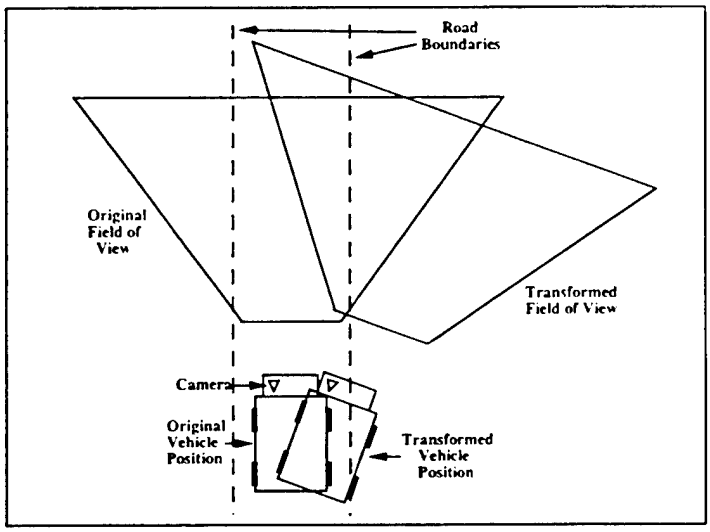


Figure 4: An aerial view of the vehicle at two different positions, with the corresponding sensor fields of view. To simulate the image transformation that would result from such a change in position and orientation of the vehicle, the overlap between the two field of view trapezoids is computed and used to direct resampling of the original image.

3.3. Extrapolating Missing Pixels

The less than complete overlap between the trapezoids of Figure 4 illustrates the need for one additional step in the image transformation scheme. The extra step involves determining values for pixels which have no corresponding pixel in the original image. Consider the transformation illustrated in Figure 5. To make it appear that the vehicle is situated one meter to the right of its position in the original image requires not only shifting pixels in the original image to the left, but also filling in the unknown pixels along the right edge. Notice the number of pixels per row whose value needs to be extrapolated is greater near the bottom of the image than at the top. This is because the one meter of unknown ground plane to the right of the visible boundary in the original image covers more pixels at the bottom than at the top. We have experimented with two techniques for extrapolating values for these unknown pixels (See Figure 6).

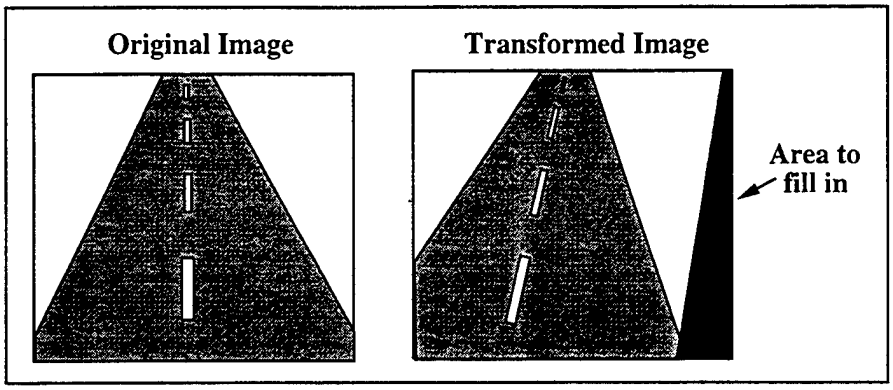


Figure 5: A schematic example of an original image, and a transformed image in which the vehicle appears one meter to the right of its initial position. The black region on the right of the transformed image corresponds to an unseen area in the original image. These pixels must be extrapolated from the information in the original image.

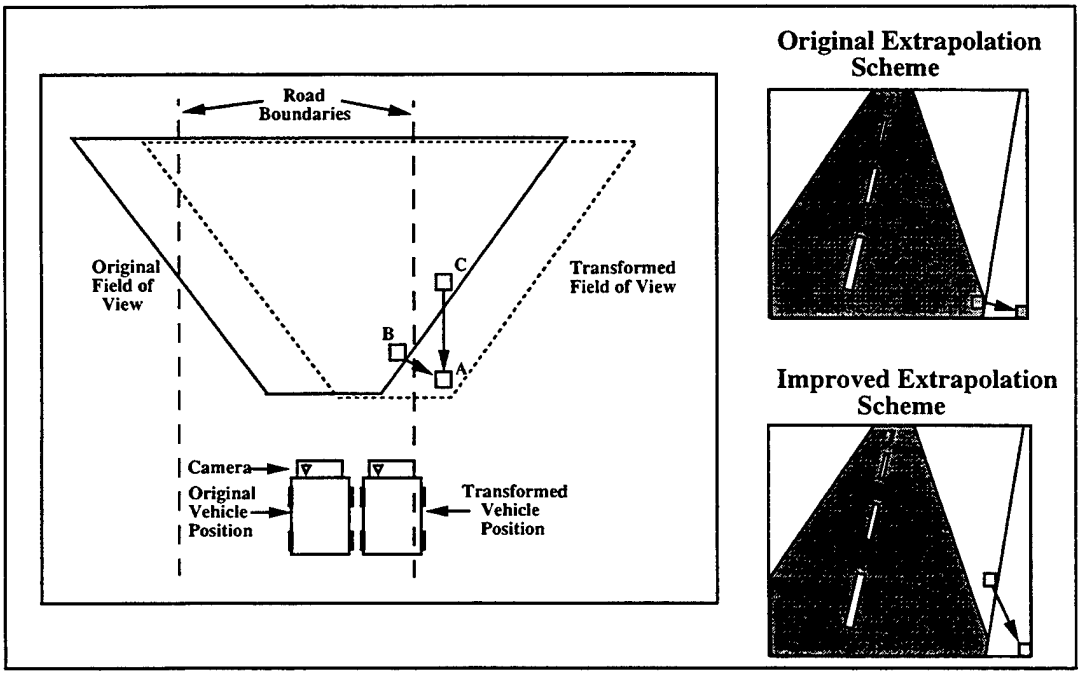


Figure 6: An aerial view (left) and image-based view (right) of the two techniques used to extrapolate the values for unknown pixels. See text for explanation.

In the first technique, to determine the value for a pixel that projects to the ground plane at point A in the transformed image, the closest ground plane point in the original viewing trapezoid (point B) is found. This point is then back-projected into the original image to find the appropriate pixel to sample. The image in the top right shows the sampling performed to fill in the missing pixel using this extrapolation scheme. The problem with this technique is that it results in the "smearing" of the image approximately along rows of the image, as illustrated in the middle image of Figure 7. In this figure, the leftmost image represents an actual reduced resolution image of a two-lane road coming from the camera. Notice the painted lines delineating the center and right boundaries of the lane. The middle image shows the original image transformed to make it appear that the

vehicle is one meter to the right of its original position using the extrapolation technique described above. The line down the right side of the road can be seen smearing to the right where it intersects the border of the original image. Because the length of this smear is highly correlated with the correct steering direction, the network learns to depend on the size of this smear to predict the correct steering direction. When driving on its own however, this lateral smearing of features is not present, so the network performs poorly.

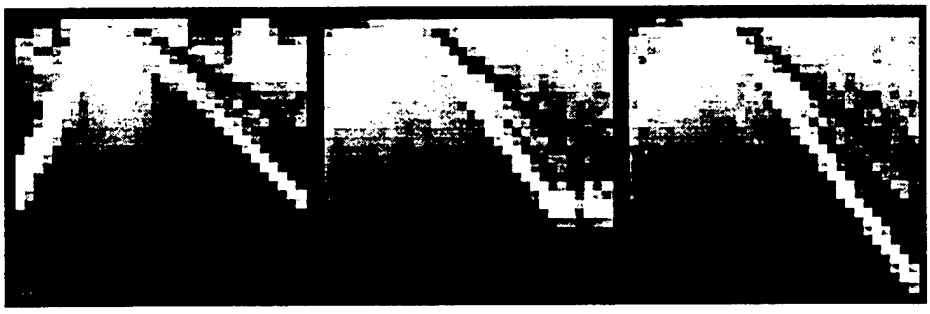


Figure 7: Three reduced resolution images of a two-lane road with lines painted down the middle and right side. The left image is the original coming directly from the camera. The middle image was created by shifting the original image to make it appear the vehicle was situated one meter to the right of its original position using the first extrapolation technique described in the text. The right image shows the same shift of the original image, but using the more realistic extrapolation technique.

To eliminate this artifact of the transformation process, we implemented a more realistic extrapolation technique which relies on the fact that interesting features (like road edges and painted lane markers) normally run parallel to the road, and hence parallel to the vehicle's current direction. With this assumption, to extrapolate a value for the unknown pixel A in Figure 6, the appropriate ground plane point to sample from the original image's viewing trapezoid is not the closest point (point B), but the nearest point in the original image's viewing trapezoid along the line that runs through point A and is parallel to the vehicle's original heading (point C).

The effect this improved extrapolation technique has on the transformed image can be seen schematically in the bottom image on the right of Figure 6. This technique results in extrapolation along the line connecting a missing pixel to the vanishing point, as illustrated in the lower right image. The realism advantage this extrapolation technique has over the previous scheme can be seen by comparing the image on the right of Figure 7 with the middle image. The line delineating the right side of the lane, which was unrealistically smeared using the previous method, is smoothly extended in the image on the right, which was created by shifting the original image by the same amount as in the middle image, but using the improved extrapolation method.

The improved transformation scheme certainly makes the transformed images look more realistic, but to test whether it improves the network's driving performance, we did the following experiment. We first collected actual two-lane road images like the one shown on the left side of Figure 6 along with the direction the driver was steering when the images were taken. We then trained two networks on this set of images. The first network was trained using the naive transformation scheme and the second using the improved transformation scheme. The magnitude of the shifts and rotations, along with the buffering scheme used in the training process are described in detail below. The networks were then tested on a disjoint set of real two-lane road images, and the steering direction dictated by the networks was compared with the person's steering direction on those images. The network trained using the more realistic transformation scheme exhibited 37% less steering error on the 100 test images than the network trained using the naive transformation scheme. In more detail, the amount of steering error a network produces is measured as the distance, in number of units (i.e. neurons), between the peak of the network's "hill" of activation in the output vector and the "correct" position, in this case the direction the person was actually steering in. This steering error measurement is illustrated in Figure 8. In this case, the network trained with the naive transformation technique had an average steering error across the 100 test images of 3.5 units, while the network trained with the realistic transformations technique had an average steering error of only 2.2 units.

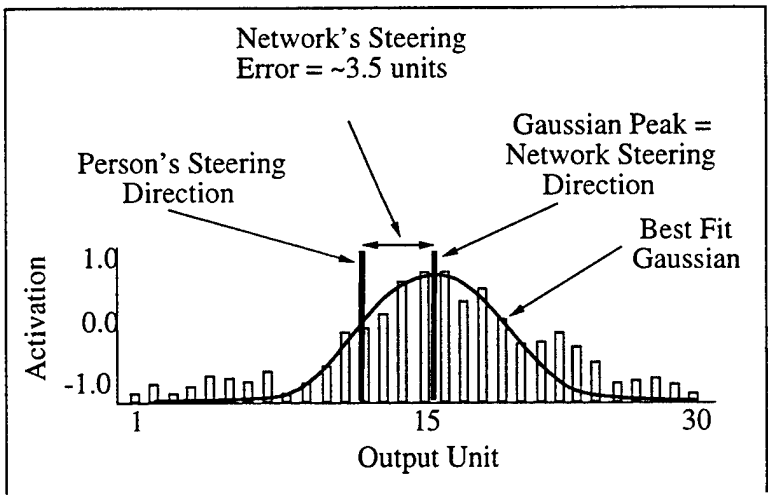


Figure 8: To calculate a network's steering error, the best fit gaussian is found to the network's output activation profile. The distance between the peak of the best fit gaussian and the position in the output vector representing the reference steering direction (in this case the person's steering direction) is calculated. This distance, measured in units or neurons between the two positions, is defined to be the network's steering error.

3.4. Transforming the Steering Direction

As important as the technique for transforming the input images is the method used to determine the correct steering direction for each of the transformed images. The correct steering direction as dictated by the driver for the original image must be altered for each of the transformed images to account for the altered vehicle placement. This is done using a simple model called pure pursuit steering [15]. In the pure pursuit model, the "correct" steering direction is the one that will bring the vehicle to a desired location (usually the center of the road) a fixed distance ahead. The idea underlying pure pursuit steering is illustrated in Figure 9. With the vehicle at position A, driving for a predetermined distance along the person's current steering arc would bring the vehicle to a "target" point T, which is assumed to be in the center of the road.

After transforming the image with a horizontal shift s and rotation θ to make it appear that the vehicle is at point B, the appropriate steering direction according to the pure pursuit model would also bring the vehicle to the target point T. Mathematically, the formula to compute the radius of the steering arc that will take the vehicle from point B to point T is

$$r = \frac{l^2 + d^2}{2d}$$

where r is the steering radius l is the look-ahead distance, and d is the distance from point T the vehicle would end up at if driven straight ahead from point B for distance l. The displacement d can be determined using the following formula:

$$d = \cos\theta(d_p + s + l\tan\theta)$$

where d_p is the distance from point T the vehicle would end up if it drove straight ahead from point A for the look-ahead distance l, s is the horizontal distance from point A to B, and θ is the vehicle rotation from point A to B. The quantity d_p can be calculated using the following equation:

$$d_p = r_p - \sqrt{r_p^2 - l^2}$$

where d_p is the radius of the arc the driver was steering along when the image was taken.

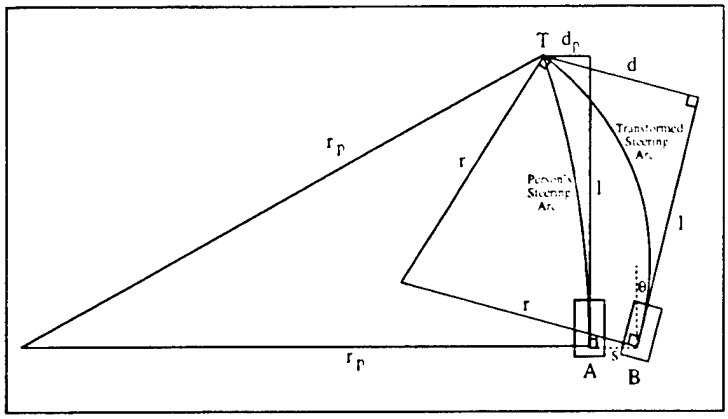


Figure 9: Illustration of the "pure pursuit" model of steering. See text for explanation.

The only remaining unspecified parameter in the pure pursuit model is *l*, the distance ahead of the vehicle to select a point to steer towards. Empirically, I have found that over the speed range of 5 to 55 mph., accurate and stable vehicle control can be achieved using the following rule: look ahead the distance the vehicle will travel in 2 to 3 seconds.

Interestingly, with this empirically determined rule for choosing the look-ahead distance, the pure pursuit model of steering is a fairly good approximation to how people actually steer. Reid, Solowka and Billing [11] found that at 50km/h, human subjects responded to a 1m lateral vehicle displacement with a steering radius ranging from 511 to 1194m. With a look-ahead equal to the distance the vehicle will travel in 2.3 seconds, the pure pursuit model dictates a steering radius of 594m, within the range of human responses. Similarly, human subjects reacted to a 1-degree heading error relative to the current road direction with a steering radius ranging from 719 to 970m. Again using the 2 to 3 second travel distance for look-ahead, the pure pursuit steering model's dictated radius of 945m falls within the range of human responses.

Like the image transformation scheme, the steering direction transformation technique uses a simple model to determine how a change in the vehicle's position and/or orientation would affect the situation. In the image transformation scheme, a planar world hypothesis and rules of perspective projection are used to determine how changing the vehicle's position and/or orientation would affect the sensor image of the scene ahead of the vehicle. In the steering direction transformation technique, a model of how people drive is used to determine how a particular vehicle transformation should alter the correct steering direction. In both cases, the transformation techniques are independent of the driving situation. The person could be driving on a single lane dirt road or a multi lane highway: the transformation techniques would be the same.

Anthropomorphically speaking, transforming the sensor image to create more training images is equivalent to telling the network, "I don't know what features in the image are important for determining the correct direction to steer, but whatever they are, here are some other positions and orientations in which you may see them ." Similarly, the technique for transforming the steering direction for each of these new training images is equivalent to telling the network "whatever the important features are, if you see them in this new position and orientation, here is how your response should change." Because it does not rely on a strong model of what important image features look like, but instead acquires this knowledge through training, the system is able to drive in a wide variety of circumstances, as will be seen later in the chapter.

These weak models are enough to solve the two problems associated with training in real time on sensor data. Specifically, using transformed training patterns allows the network to learn how to recover from driving mistakes that it would not otherwise encounter as the person drives. Also, overtraining on repetitive images is less of a problem, since the transformed training exemplars maintain variety in the training set.

3.5. Adding Diversity Through Buffering

As additional insurance against the effects of repetitive exemplars, the training set diversity is further increased by maintaining a buffer of previously encountered training patterns. When new training patterns are acquired through digitizing and transforming the current sensor image, they are added to the buffer, while older patterns are removed. We have experimented with four techniques for determining which patterns to replace. The first is to replace the oldest patterns first. Using this scheme, the training pattern buffer represents a history of the driving situations encountered recently. But if the driving situation remains unchanged for a period of time, such as during an extended right turn, the buffer will loose its diversity and become filled with right turn patterns. The second technique is to randomly choose old patterns to be replaced by new ones. Using this technique, the laws of probability help ensure somewhat more diversity than the oldest pattern replacement scheme, but the buffer will still become biased during monotonous stretches.

The next solution we developed to encourage diversity in the training was to replace those patterns on which the network was making the lowest error, as measured by the sum squared difference between the network's output and the desired output. The idea was to eliminate the patterns the network was performing best on, and leave in the training set those images the network was still having trouble with. The problem with this technique results from the fact that the human driver doesn't always steer in the correct direction. Occasionally he may have a lapse of attention for a moment and steer in an incorrect direction for the current situation. If a training exemplar was collected during this momentary lapse, under this replacement scheme it will remain there in the training buffer for a long time, since the network will have trouble outputting a steering response to match the person's incorrect steering command. In fact, using this replacement technique, the only way the pattern would be removed from the training set would be if the network learned to duplicate the incorrect steering response, obviously not a desired outcome. I considered replacing both the patterns with the lowest error *and* the patterns with the highest error, but decided against it since high network error on a pattern might also result on novel input image with a correct response associated with it. A better method to eliminate this problem is to add a random replacement probability to all patterns in the training buffer. This ensured that even if the network never learns to produce the same steering response as the person on an image, that image will eventually be eliminated from the training set.

While this augmented lowest-error-replacement technique did a reasonable job of maintaining diversity in the training set, we found a more straightforward way of accomplishing the same result. To make sure the buffer of training patterns does not become biased towards one steering direction, we add a constraint to ensure that the mean steering direction of all the patterns in the buffer is as close to straight ahead as possible. When choosing the pattern to replace, I select the pattern whose replacement will bring the average steering direction closest to straight. For instance, if the training pattern buffer had more right turns than left, and a left turn image was just collected, one of the right turn images in the buffer would be chosen for replacement to move the average steering direction towards straight ahead. If the buffer already had a straight-ahead average steering direction, then an old pattern, requiring a similar steering direction for the new one, would be replaced in order to maintain the buffer's unbiased nature. By actively compensating for steering bias in the training buffer, the network never learns to consistently favor one steering direction over another. This active bias compensation is a way to build into the network a known constraint about steering: in the long run, right and left turns occur with equal frequency.

3.6. Training Details

The final details required to specify the training on-the-fly process are the number and magnitude of transformations to use for training the network. The following quantities have been determined empirically to provide sufficient diversity to allow networks to learn to drive in a wide variety of situations. The original sensor image is shifted and rotated 14 times using the technique describe above to create 14 training exemplars. The size of the shift for each of the transformed exemplars is chosen randomly from the range -0.6 to +0.6 meters, and the amount of rotation is chosen from the range -6.0 to +6.0 degrees. In the image formed by the camera on the Navlab, which has a 42-degree horizontal field of view, an image with a maximum shift of 0.6m results in the

road shifting approximately 1/3 of the way across the input image at the bottom.

Before the randomly selected shift and rotation is performed on the original image, the steering direction that would be appropriate for the resulting transformed image is computed using the formulas given above. If the resulting steering direction is sharper than the sharpest turn representable by the network's output (usually a turn with a 20m radius), then the transformation is disallowed and a new shift distance and rotation magnitude are randomly chosen. By eliminating extreme and unlikely conditions from the training set, such as when the road is shifted far to the right and vehicle is heading sharply to the left, the network is able to devote more of its representation capability to handling plausible scenarios.

The 14 transformed training patterns, along with the single pattern created by pairing the current sensor image with the current steering direction, are inserted into the buffer of 200 patterns using the replacement strategy described above. After this replacement process, one forward and one backward pass of the back-propagation algorithm is performed on the 200 exemplars to update the network's weights, using a learning rate of 0.01 and a momentum of 0.8. The entire process is then repeated. Each cycle requires approximately 2.5 seconds on the three Sun Sparcstations onboard the vehicle. One of the Sparcstation performs the sensor image acquisition and preprocessing, the second implements the neural network simulation, and the third takes care of communicating with the vehicle controller and displaying system parameters for the human observer. The network requires approximately 100 iterations through this digitize-replace-train cycle to learn to drive in the domains that have been tested. At 2.5 seconds per cycle, training takes approximately four minutes of human driving over a sample stretch of road. During the training phase, the person drives at approximately the speed at which the network will be tested, which ranges from 5 to 55 mph.

4. Performance Improvement Using Transformations

The performance advantage this technique of transforming and buffering training patterns offers over the more naive methods of training on real sensor data is illustrated in Figure 10. This graph shows the vehicle's displacement from the road center, measured as three different networks drove at 4 m.p.h. over a 100m section of a single lane paved bike path which included a straight stretch and turns to the left and right. The three networks were trained over a 150m stretch of the path which was disjoint from the test section and which ended in an extended right turn.

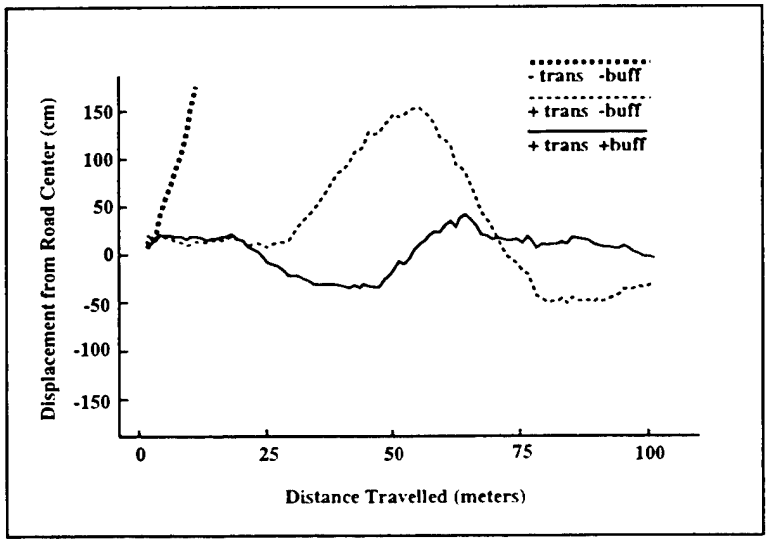


Figure 10: Vehicle displacement from the road center as the Navlab was driven by networks trained using three different techniques.

The first network, labeled "-trans -buff," was trained using just the images coming from the video camera. That

is, during the training phase, an image was digitized from the camera and fed into the network. One forward and backward pass of back-propagation was performed on that training exemplar, and then the process was repeated. The second network, labeled "+trans -buff," was trained using the following technique. An image was digitized from the camera and then transformed 14 times to create 15 new training patterns as described above. A forward and backwards pass of back-propagation was then performed on each of these 15 training patterns and then the process was repeated. The third network, labeled "+trans +buff," was trained using the same transformation scheme as the second network, but with the addition of the image buffering technique described above to prevent overtraining on recent images.

Note that all three networks were presented with the same number of images. The transformation and buffering schemes did not influence the quantity of data the networks were trained on, only its distribution. The "-trans -buff" network was trained on closely spaced actual video images. The "+trans -buff" network was presented with 15 times fewer actual images, but its training set also contained 14 transformed images for every "real" one. The "+trans +buff" network collected even fewer live images, since it performed a forward and backward pass through its buffer of 200 patterns before digitizing a new one.

The accuracy of each of the three networks was determined by manually measuring the vehicle's lateral displacement relative to the road center as each network drove. The network trained on only the current video image quickly drove off the right side of the road, as indicated by its rapidly increasing displacement from the road center. The problem was that the network overlearned the right turn at the end of training and became biased towards turning right. Because of the increased diversity provided by the image transformation scheme, the second network performed much better than the first. It was able to follow the entire test stretch of road. However, it still had a tendency to steer too much to the right, as illustrated in the graph by the vehicle's positive displacement over most of the test run. In fact, the mean position of the vehicle was 28.9cm right of the road center during the test. The variability of the errors made by this network was also quite large, as illustrated by the wide range of vehicle displacement in the "+trans -buff" graph. Quantitatively, the standard deviation of this network's displacement was 62.7cm.

The addition of buffering previously encountered training patterns eliminated the right bias in the third network, and also greatly reduced the magnitude of the vehicle's displacement from the road center, as evidenced by the "+trans +buff" graph. While the third network drove, the average position of the vehicle was 2.7cm right of center, with a standard deviation of only 14.8cm. This represents a 423% improvement in driving accuracy.

A separate test was performed to compare the steering accuracy of the network trained using both transformations and buffering with the steering accuracy of a human driver. This test was performed over the same stretch of road as the previous one; however, the road was less obscured by fallen leaves in this test, resulting in better network performance. Over three runs, with the network driving at 5 miles per hour along the 100 m test section of road, the average position of the vehicle was 1.6 cm right of center, with a standard deviation of 7.2 cm. Under human control, the average position of the vehicle was 4.0 cm right of center, with a standard deviation of 5.47 cm. The average distance the vehicle was from the road center while the person drove was 5.70 cm. It appears that the human driver, while more consistent than the network, had an inaccurate estimate of the vehicle's centerline, and therefore drove slightly to the right of the road center. Studies of human driving performance have found similar steady state errors and variances in vehicle lateral position. Blaauw [2] found consistent displacements of up to 7 cm were not uncommon when people drove on highways. Also for highway driving, Blaauw reports standard deviations in lateral error up to 16.6 cm.

5. Results and Comparison

The competence of the ALVINN system is also demonstrated by the range of situations in which it has successfully driven.

The training on-the-fly scheme gives ALVINN a flexibility which is novel among autonomous navigation systems. It has allowed individual networks to be successfully trained to drive in a variety of situations, including a single-lane dirt access road, a single-lane paved bicycle path, a two-lane suburban neighborhood street, and a lined two-lane highway (See Figure 11). Using other sensor modalities as input, including laser range images and laser reflectance images, individual ALVINN networks have been trained to follow roads in total darkness,

to avoid collisions in obstacle rich environments, and to follow alongside railroad tracks. ALVINN networks have driven without intervention for distances of up to 22 miles. In addition, since determining the steering direction from the input image merely involves a forward sweep through the network, the system is able to process 15 images per second, allowing it to drive at up to 55 miles per hour. This is over four times faster than any other sensor-based autonomous system, using the same processing hardware, has driven the Navlab[5][10].

Figure 11: Video images taken on three of the road types ALVINN modules have been trained to handle. They are, from left to right, a single-lane dirt access road, a single-lane paved bicycle path, and a lined two-lane highway.

The level of flexibility across driving situations exhibited by ALVINN would be difficult to achieve without learning. It would require the programmer to 1) determine what features are important for the particular driving domain, 2) program detectors (using statistical or symbolic techniques) for finding these important features and 3) develop an algorithm for determining which direction to steer from the location of the detected features. As a result, while hand-programmed systems have been developed to drive in some of the individual domains ALVINN can handle [5][10][12][8], none have duplicated ALVINN's flexibility.

ALVINN is able to *learn* for each new domain what image features are important, how to detect them and how to use their positions to steer the vehicle. Analysis of the hidden unit representations developed in different driving situations shows that the network forms detectors for the image features which correlate with the correct steering direction. When trained on multi-lane roads, the network develops hidden unit feature detectors for the lines painted on the road, while in single-lane driving situations, the detectors developed are sensitive to road edges and road-shaped regions of similar intensity in the image. For a more detailed analysis of ALVINN's internal representations see [14][13].

This ability to use arbitrary image features can be problematic. This was the case when ALVINN was trained to drive on a poorly defined dirt road with a distinct ditch on its right side. The network had no problem learning and then driving autonomously in one direction; but when driving the other way, the network was erratic, swerving from one side of the road to the other. After analyzing the network's hidden representation, the reason for its difficulty became clear. Because of the poor distinction between the road and the non-road, the network had developed only weak detectors for the road itself and instead relied heavily on the position of the ditch to determine the direction to steer. When tested in the opposite direction, the network was able to keep the vehicle on the road using its weak road detectors but was unstable because the ditch it had learned to look for on the right side was now on the left. Individual ALVINN networks have a tendency to rely on *any* image feature consistently correlated with the correct steering direction. Therefore, it is important to expose them to a wide enough variety of situations during training so as to minimize the effects of transient image features.

On the other hand, experience has shown that it is more efficient to train several domain-specific networks for circumstances like one-lane vs. two-lane driving, instead of training a single network for all situations. To prevent this network specificity from reducing ALVINN's generality, we are currently implementing connectionist and non-connectionist techniques for combining networks trained for different driving situations. Using a simple rule-based priority system similar to the subsumption architecture [3], we have combined a road following network and an obstacle avoidance network. The road following network uses video camera input to follow a single-lane road. The obstacle avoidance network uses laser rangefinder images as input. It is trained

to swerve appropriately to prevent a collision when confronted with obstacles and to drive straight when the terrain ahead is free of obstructions. The arbitration rule gives priority to the road following network when determining the steering direction, except when the obstacle avoidance network outputs a sharp steering command. In this case, the urgency of avoiding an imminent collision takes precedence over road following and the steering direction is determined by the obstacle avoidance network. Together, the two networks and the arbitration rule comprise a system capable of staying on the road and swerving to prevent collisions.

To facilitate other rule-based arbitration techniques, we have added to ALVINN a non-connectionist module which maintains the vehicle's position on a map [6]. Knowing its map position allows ALVINN to use arbitration rules such as "when on a stretch of two lane highway, rely primarily on the two-lane highway network." This symbolic mapping module also allows ALVINN to make high-level, goal-oriented decisions, such as which way to turn at intersections and when to stop at a predetermined destination.

Finally, we are experimenting with connectionist techniques such as the task decomposition architecture [16] and the meta-pi architecture [9] for combining networks more seamlessly than is possible with symbolic rules. These connectionist arbitration techniques will enable ALVINN to combine outputs from networks trained to perform the same task using different sensor modalities and to decide when a new expert must be trained to handle the current situation.

6. Discussion

A truly autonomous mobile vehicle must cope with a wide variety of driving situations and environmental conditions. As a result, it is crucial that an autonomous navigation system possess the ability to adapt to novel domains. Supervised training of a connectionist network is one means of achieving this adaptability. But teaching an artificial neural network to drive based on a person's driving behavior presents a number of challenges. Prominent among these is the need to maintain sufficient variety in the training set to ensure that the network develops a sufficiently general representation of the task. Two characteristics of real sensor data collected as a person drives make training set variety difficult to maintain; these characteristics are temporal correlations and the limited range of situations encountered. Extended intervals of nearly identical sensor input can bias a network's internal representation and reduce later driving accuracy. The human trainer's high degree of driving accuracy severely restricts the variety of situations covered by the raw sensor data.

The techniques for training "on-the-fly" described in this chapter solve these difficulties. The key idea underlying training on-the-fly is that a model of the process generating the live training data can be used to augment the training set with additional realistic patterns. By modeling both the imaging process and the steering behavior of the human driver, training on-the-fly generates patterns with sufficient variety to allow artificial neural networks to learn a robust representation of individual driving domains. The resulting networks are capable of driving accurately in a wide range of situations.

Bibliography

[1] G. Hinton K. Shikano A. Waibel, T. Hanazawa and K. Lang. Phoneme recognition: Neural networks vs. hidden markov models. In *Proceedings from Int. Conf. on Acoustics Speech and Signal Processing.* New York, New York. 1988.

[2] G.J. Blaauw. Driving experience and task demands in simulator and instrumented car: A validation study. *Human Factors.* 24:473-486. 1982.

[3] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation.* RA-2(1):14--23. 1986.

[4] G.W. Cottrell. Extracting features from faces using compression networks: Face identity emotion and gender recognition using holons. In *Connectionist Models: Proceedings of the 1990 Summer School.* San Mateo CA. Morgan Kaufmann Publishers. 1990.

[5] J.D. Crisman and C.E. Thorpe. Color vision for road following. In C.E. Thorpe, editor, *Vision and*

Navigation: The CMU Navlab. Kluwer Academic Publishers, Boston Massachusetts. 1990.

[6] J. Gowdy, D.A. Pomerleau and C.E. Thorpe. Combining artificial neural networks and symbolic processing for autonomous robot guidance. *Engineering Applications of Artificial Intelligence.* 4(4):279--285. 1991.

[7] G.E. Hinton D.E. Rumelhart and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, Eds. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition.* Volume 1. Bradford Books/MIT Press. Cambridge Massachusetts, 1986.

[8] E.D. Dickmanns and A. Zapp. Autonomous high speed road vehicle guidance by computer vision. In *Proceedings of the 10th World Congress on Automatic Control.* Volume 4. Munich, Germany. 1987.

[9] J.B. Hampshire and A.H. Waibel. *The meta-pi network: Building distributed knowledge representations for robust pattern recognition.* Technical Report CMU-CS-89-166-R. Carnegie Mellon University, Pittsburgh PA. 1989.

[10] K. Kluge and C.E. Thorpe. Explicit models for robot road following. In C.E. Thorpe, editor, *Vision and Navigation: The CMU Navlab.* Kluwer Academic Publishers, Boston MA. 1990.

[11] \E.N. Solowka, L.D. Reid and A.M. Billing. A systematic study of driver steering behaviour. *Ergonomics.* 24:447-462. 1981.

[12] K.D. Gremban M.A. Turk, D.G. Morgenthaler and M. Marra. Vits -- a vision system for autonomous land vehicle navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Vol. 10. 1988.

[13] D.A. Pomerleau, editor. *Neural network perception for mobile robot guidance.* Kluwer Academic Publishing. 1993.

[14] D.A. Pomerleau and D.S. Touretzky. Understanding neural network internal representations through hidden unit sensitivity analysis. In *Proceedings of the International Conference on Intelligent Autonomous Systems.* IOS Publishers, Amsterdam. 1993.

[15] C. Thorpe, H. Moravec, W. Whittaker, R. Wallace, A. Stentz and T. Kanade. First results in robot road-following. *Proceedings of Int. Joint Conf. on Artificial Intelligence.* 1985.

[16] \A.G. Barto R.A. Jacobs, M.I. Jordan. *Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks.* Technical Report 90-27, Univ. of Massachusetts Computer and Information Science Department. 1990.

[17] J.S. Denker D. Henderson R.E. Howard W. Hubbard Y. LeCun, B. Boser and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation,* 1(4). 1989.

Vision-Based Neural Network Road and Intersection Detection

1. Introduction

Much progress has been made toward solving the autonomous lane-keeping problem. Systems which can drive robot vehicles at high speeds for long distances have been demonstrated. Some systems use road models to determine where lane markings are expected[2][6][7], while others are based on artificial neural networks which learn the salient features required for driving on a particular road type[4][5][10].

The current challenge for vision-based navigation researchers is to build on the performance of the already developed lane-keeping systems, adding the ability to perform higher-level driving tasks. These tasks include actions such as lane changing, localization, and intersection detection and navigation. This paper examines the task of road and intersection detection and navigation.

The work presented in this chapter is based on the ALVINN neural network-based lane-keeping system. The extended ALVINN system, one which is capable of detecting roads and intersection, is called ALVINN VC (VC for Virtual Camera). ALVINN VC uses the robust road detection and confidence measurement capability of the core ALVINN system along with an artificial imaging sensor to reliably detect road segments which occur at locations other than immediately in front of the vehicle and the camera.

The imaging sensor that ALVINN VC uses is called a **virtual camera** and is described in detail in Section 2. Virtual cameras are the fundamental tool upon which the techniques presented in this paper are based. They provide a mechanism for determining the appropriateness of vehicle actions, but do not compromise the robust driving performance of the core ALVINN system.

2. The virtual camera

A virtual camera is simply an imaging sensor which can be placed at any location and orientation in the world reference frame. It creates artificial images using actual pixels imaged by a real camera that have been projected onto some world model. By knowing the location of both the actual and virtual camera, and by assuming a flat world model, accurate image reconstructions, called virtual images, can be created from the virtual camera location. Virtual camera views have been used by ALVINN VC to successfully navigate on all road types which the original ALVINN system performed.

An interesting issue that is a general theme of this paper is the ability of virtual cameras to merge neural systems with symbolic ones. Virtual cameras impose a geometric model on the neural system. In our case, the model is not a feature in an image, but rather a canonical image viewpoint which ALVINN VC can interpret. To ALVINN VC, the virtual camera is a sensing device. It is ALVINN VC's only link to the world in which it operates. ALVINN VC doesn't care where the virtual camera is located, only that it is producing images which are similar to those on which it was trained and can thus be used to locate the road. This interpretation may seem to trivialize ALVINN VC's functionality, but in reality, finding the road is what ALVINN VC is designed to do best. The virtual camera insures that the system gets images which will let it do its job to the best of its ability. The details of creating appropriate virtual camera locations and interpreting the resulting output are left to other, higher-level modules. So in essence, the virtual camera imposes a geometric model on ALVINN VC without it knowing, or even caring, about it. Used in conjunction with higher-level modules, the model allows ALVINN VC to exhibit goal-directed, intelligent behavior without compromising system performance.

3. Detection philosophy

There are three principles upon which road and intersection detection and navigation systems should be based. They are:

1. Detection and navigation should be data (image) driven.

2. Detection is signaled by the presence of features.

3. Road junctions should be traversed by actively tracking the road or intersection branch.

These principles and their relationship to ALVINN VC, as well as other road and intersection detection systems, are examined in greater detail in [3].

ALVINN VC uses a priori knowledge that specifies where and when appropriate virtual cameras should be created. The cameras are created relative to the vehicle and creation does not coincide with when the intersection is actually located, but rather somewhere before it occurs. Instead of "Now we are at the intersection, so look for its branches," the system deals with information like "Start looking for a single lane road." The virtual camera's location, and the network associated with each, is dependent upon the type of road that is expected to be encountered. When the road or intersection to be detected is present, the virtual cameras image it in a way that is meaningful to the system's neural networks. By continually monitoring the network's confidence for each virtual camera, the system can determine when the road or intersection is present.

ALVINN VC currently adheres to the first two principles of road junction detection and traversal mentioned earlier. Also, methods are under development that will allow ALVINN VC to actively track roads using a combination of active camera control and intelligently placed virtual cameras.

4. Other systems

Several other groups have built systems [1][8][10][11] to study the road and intersection detection problem. Many of them have adopted a data directed approach, but many also rely on the absence of features rather than the presence of them to indicate when a road or intersection is present. Few use active camera control. A complete summary of these system can be found in [3].

5. Experimental results

A series of experiments was conducted to assess the usefulness of virtual cameras for autonomously detecting roads and intersections. All of the experiments described were performed on the Navlab 2. The experimental site was a single lane paved path near the Carnegie Mellon campus. The path was unlined with grass on either side and was 3.1 meters wide.

The first experiment was very simple and designed to assess the basic ability of virtual cameras to create images which were usable by the system. For this, a single virtual view was used to detect an upcoming road and to navigate onto it. The second experiment was more challenging - virtual cameras were used to not only keep the vehicle on the road, but to also detect an upcoming 'Y' intersection. After the intersection was detected, the system used higher level information to choose the appropriate fork to follow.

5.1. Road detection experiment

This experiment was designed to test the system's robustness for detecting and navigating onto roads. In this

experiment, the system had information that the vehicle was approaching a road perpendicularly. Its job was to detect the road, drive the vehicle onto it, and then continue operating in a normal autonomous driving mode. The vehicle was not on another road as it approached the road to be detected. This scenario corresponds to ending a cross-country navigation mission and acquiring a road to begin autonomous road following.

Initially, the vehicle was positioned approximately 35 meters off of the road which was to be detected, and aligned perpendicularly to it. A virtual view was created that was rotated 90 degrees from the direction of vehicle travel. This view was placed 20 meters in front of the vehicle (Figure 1).

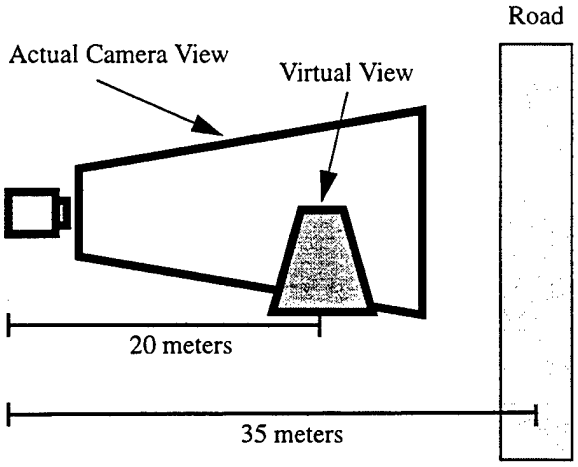


Figure 1: Virtual view for detecting a perpendicular road.

The vehicle was instructed to move along its current heading until the system detected the road. At this point, the system instructed the vehicle to turn appropriately based on the point specified by the neural network. Once the system had aligned the vehicle sufficiently with the road, it was instructed to begin road following.

5.2. Road detection

The ability to detect the upcoming road was the first and most important requirement of the system. To accomplish this, every 0.3 second as the vehicle approached the road (at a speed of about 5 m.p.h.), a virtual image was created and passed to ALVINN VC's neural network. The network produced an output vector, interpreted as a point on the road to drive over, and a confidence value using the Input Reconstruction Reliability Estimation (IRRE) metric. This metric is described in greater detail in Section 5.3.. To determine when the system had actually located the road, the IRRE metric was monitored. When this metric increased above a user-defined threshold value, which was typically 0.8 (out of 1.0), ALVINN VC reported that it had located the road.

5.3. Application of IRRE to road detection

IRRE is a measure of the familiarity of the input image to the neural network. In IRRE, the network's internal representation is used to reconstruct, on a set of output units, the input pattern being presented. The more closely the reconstructed input matches the actual input, the more familiar the input and hence, the more reliable the network's response.

The network is trained using backpropagation to both produce the correct steering response on the steering output units and to reconstruct the input image as accurately as possible on the reconstruction outputs.

During testing, images are presented to the network and activation is propagated forward to produce a steering response and a reconstructed input image. The reliability of the steering response is estimated by computing the correlation coefficient between the activation levels of units in the actual input image and the reconstructed input image. The higher the correlation between the two images, the more reliable the network's steering response is estimated to be[10].

Using the IRRE metric to indicate when roads are present in the input virtual image assumes that it will be low for images which do not contain roads and distinctly higher for those that do. For this assumption to hold, two things must occur. First, the system's neural network must not be able to accurately reconstruct images which do not contain roads, leading to a low IRRE measure. Second, images created by the virtual camera when a road is present must look sufficiently similar to ones seen during training, thus leading to an accurate reconstruction and a high IRRE response.

To test these assumptions, several image were taken at various distances from the road as the vehicle approached. In each of these images, the location of the virtual camera was moved so that it imaged areas between the vehicle and the road, on the road, and past the road. Specifically, actual images were taken when the vehicle was at distances of 25, 20, 15, and 10 meters from the center of the road. Virtual camera images were created at 1 m intervals on either side of the expected road location. For example, using the actual image taken 20 meters from the road center, virtual views were created every meter between the distances of 14 meters to 29 meters.

For each actual image, virtual camera images were created at intervals similar to those specified above and given to a network previously trained to drive on the one-lane road. The output road location and the IRRE confidence metric were computed. The result of this experiment is shown in Figure 2, which shows the IRRE response with respect to the position of the virtual view for each actual image. For each actual image, the network's IRRE response clearly peaks near the expected road distance. As the virtual view moves closer to the road, the IRRE response increases, peaking when the virtual view is directly over the road. Response quickly falls again after the view passes over the road. For comparison, when ALVINN is driving on a familiar road, the IRRE response is typically between 0.70 and 0.95. The peaks in each IRRE curve actually occur about 2 meters past the actual road center. This is due to three things: a violation of the flat world assumption, errors in camera calibration, and improper initial alignment to the road.

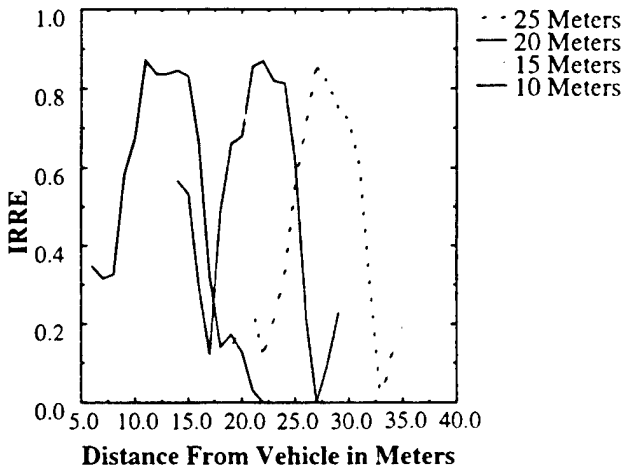


Figure 2: Input reconstruction metric for different virtual views.

Figure 2 shows that both assumptions are basically correct - the IRRE response when the network is not being presented road images is low, and the IRRE response is high when the network is being presented accurately

imaged virtual views.

The relationship between the input virtual image and the IRRE value associated with that image is better shown in Figure 3.

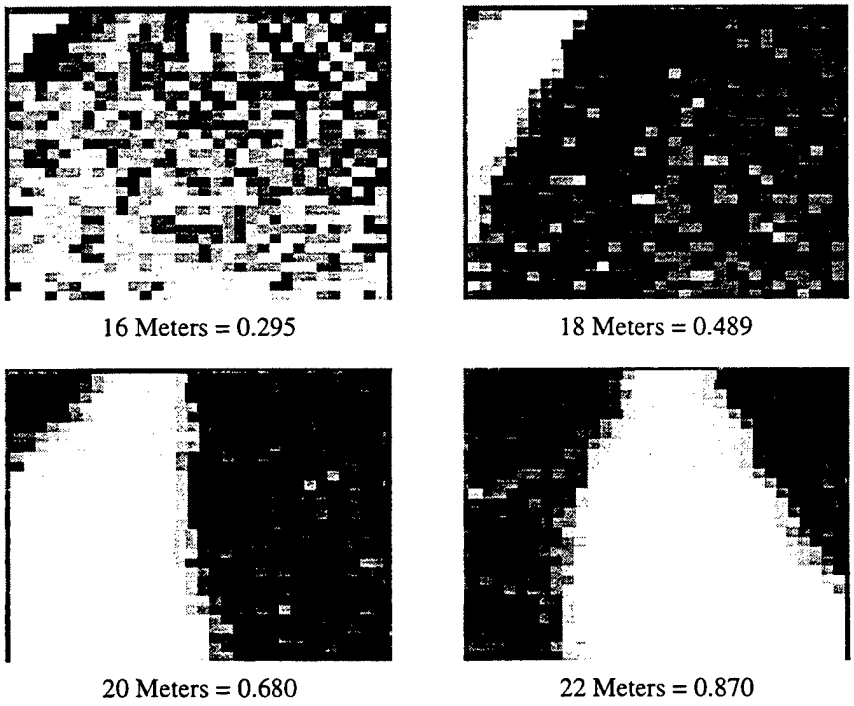


Figure 3: Virtual images at different distances in front of the vehicle and corresponding IRRE values.

The figure shows virtual images created at different distances in front of the vehicle along with the IRRE response they solicit. The images are all from an actual image that was taken when the vehicle was 20 meters from the road center. In the upper left image, the road is barely visible and, as expected, the IRRE response is very low. As the virtual view is moved forward, shown in the upper right and lower left images, it begins to image more of the road. The IRRE value increase correspondingly. The trend continues until the virtual view is centered over the road, as shown in the lower right image. At this location, the IRRE value peaks.

Each of the IRRE response curves shown in Figure 2 clearly indicate that a road is present at some distance in front of the vehicle. Because it is generally better to detect a road at a greater distance, it is desirable to know if accuracy in detection decreases as the distance from the vehicle to the road increases. Insight to this can be gained by transforming all of the curves from Figure 2 into the same reference frame. This can be done for each of the virtual views associated with a single actual image by subtracting the distance between the vehicle and the road center from the virtual camera location. This results in a coordinate system whose origin is at the center of the road. The result of transforming each of the response curves in Figure 2 into this coordinate frame is shown in Figure 4. This graph shows that detection accuracy, at least when approaching the road perpendicularly, does not degrade as distance from the road increases.

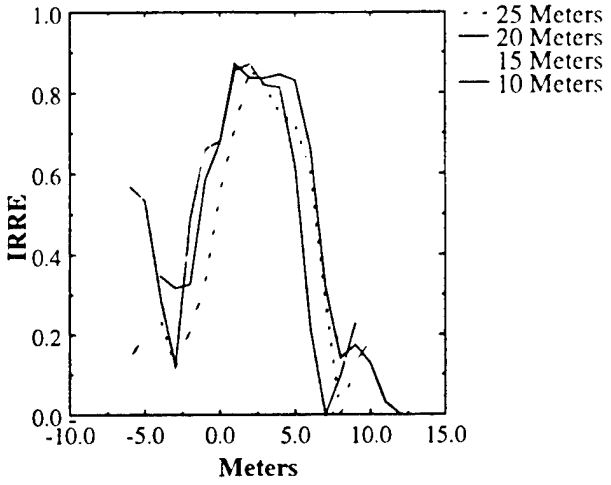


Figure 4: IRRE curves after translation to the road center.

5.4. Alignment

While testing the detection phase of the system, it became clear that the problem would not be detecting the road, but rather driving onto it after it was detected. The next sections detail two algorithms used to drive the vehicle onto the road. The algorithms are presented in increasing order of robustness. The detection method described previously was used for finding the road for each method.

5.4.1. Simple road alignment

The first algorithm that was tested for moving the vehicle onto the road was to simply drive the vehicle over the point on the road which was specified by the system. For our vehicle, this meant that the center of the rear axle would pass over the specified road point. (The center of the rear axle is the origin of the vehicle coordinate system. Our point tracking algorithm uses this point as the location on the vehicle which should follow points to be tracked.)

The point tracking algorithm was able to reliably position the vehicle over the detected road point. The problem with this approach was that the vehicle heading was not matched with the road orientation (See Figure 5). Consequently, in many cases the vehicle was not able to begin road following after it had reached the road point because the road was no longer visible in the camera's field of view. One cause of this situation is that our point tracking algorithm, pure pursuit, does not attempt to match desired and actual headings. But even if it did, the combination of the computed road point location relative to the vehicle origin and the minimum turn radius of the vehicle would prevent proper alignment to the road in some cases. Basically, the road position computed using the virtual view does not provide enough offset from the original direction of travel to allow the necessary turn to be executed. The virtual view, and, in turn, the computed road position, is constrained by the actual camera - a large portion of the virtual view must be within the actual camera's field of view in order for realistic images to be created. This result suggests that another point on the road, further along it in the desired direction of travel, is needed.

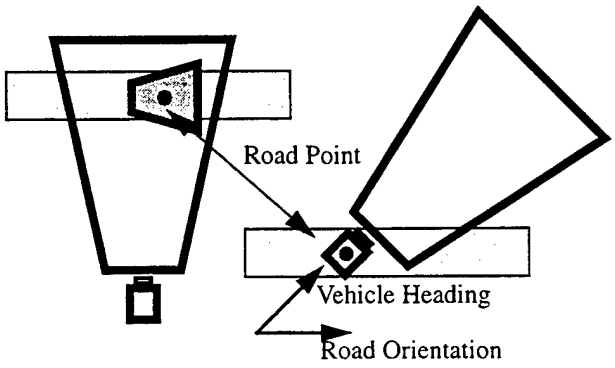


Figure 5: A case in which vehicle heading and road orientation are not matched.

5.4.2. Alignment by projecting along the road

To remedy the heading match problem encountered in the previous experiment, another point (P2) was created in addition to the network's output road location (P1). P2 was created using information about the orientation of the virtual view with respect to the vehicle. By doing this, it is assumed that the orientation of the virtual view is consistent with the expected road orientation. For example, when the virtual view is at a 90° angle with respect to the vehicle, P2 was created by projecting from P1 at an angle of 90° from the line that runs forward from the vehicle origin. P2 is assumed to be on the road. The projection distance was typically 20 meters (Figure 6).

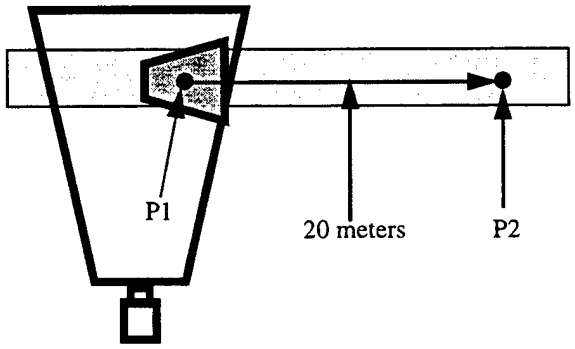


Figure 6: Definition of the second reference point P2.

Whereas the first technique computed a single arc to drive to reach the road point, this technique requires more advanced interaction with the point-tracking algorithm. Because the two points along with the vehicle location define a path to follow rather than just a point to drive over, other variables, such as the lookahead distance of the point tracker, the projection distance from P1 to P2, and the detection distance effect system performance. These parameters are discussed in detail in [3].

The selection of these parameters is not independent -- changing one will likely require changing others in order to maintain system performance. This made developing a set of consistently usable parameters very difficult. In some trials, the vehicle turned smoothly onto the road and was able to begin road following. Other times, it turned too sharply and could not locate the road at all. In still other instances, it would cross over the road in its path to reach the projected road point.

There are two main disadvantages to this approach. The first is that it is not a trivial task to develop a correct

set of point tracking parameters. Different parameters would likely be needed for every detection scenario and although it could be done, it would be a very large, brittle, and inelegant solution.

The second reason relates directly to determining P2. A large projection distance when computing P2 is desirable, but may not accurately model the road. A similar situation can happen even with small projection distances if the virtual view is not oriented exactly with the road. This occurs because ALVINN VC's neural network is trained on images which are created as if the vehicle was shifted and/or *rotated* from it true location. In the road detection scenario, this means that even if the road is not at the precise orientation defined by the virtual view, the network will respond with a high confidence. As a result, the road may not continue in the virtual view orientation and projecting to find P2 will yield a point off the road.

5.5. Intersection detection experiments

In this experiment, the goal was to drive along a single lane road, search for and detect a 'Y' intersection, and drive onto one fork or the other (Figure 7).

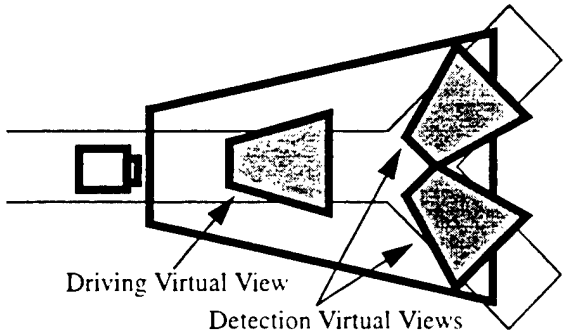


Figure 7: Virtual view configuration for the detection of a Y intersection.

The central point of this experiment was to determine if intersections could be detected by extending the work done for detecting single roads. This experiment was more difficult than the previous road detection experiments for two reasons. First, it required that the system keep the vehicle on the road and at the same time look for the intersection branches. Second, it required that the system find two road branches rather than just one. Another factor adding to the difficulty of the scenario is that the intersection lies at the crest of a small hill - each of the road segments which meet at the intersection are inclined. This means that the flat world assumption is violated.

The road that the vehicle is traveling upon as well as each of the road branches are of the same type. Virtual views were created 9 meters in front of the vehicle. The view that was used to search for the left fork was angled 25 degrees to the left of straight ahead. The one used to search for the right fork was 20 degree right of straight ahead. Because of the geometry of the situation, the IRRE threshold value, which both virtual images were required to exceed, on a single actual image was lowered to 0.70. The experiment was conducted several times, with the results from each being similar to those of the single road case. The system was able to drive the vehicle at low speeds (5mph) and detect each of the road branches. Although not as pronounced as in the single road detection case presented earlier, the system still had problems navigating onto either branch.

6. Conclusions and future work

Clearly, there is much work left to be done to robustly detect all roads and intersections. This paper presents a vision-based approach which uses the core of a robust neural network road follower to accurately detect single

lane, unlined roads. It is reasonable to assume that the detection method is directly extendable to any road type which the base neural network can learn to drive on. If this assumption is, in fact, found to be true, this system will have an advantage over other road and intersection detection systems which require the researcher to program in new detection methods when new road types are encountered.

Currently, the weak link of the system is its inability to navigate road junctions once they are found. We are investigating active camera control methods to address this problem.

Finally, the results presented were from a real, but fairly constrained environment. A robust road and intersection detection system must be able operate in more challenging environments: on typical city streets, with other cars, and with more extensive interaction with higher-level knowledge. These areas are also actively being pursued.

Bibliography

[1] J.D. Crisman. *Color Vision for the Detection of Unstructured Roads and Intersections.* Ph.D. dissertation. Carnegie Mellon University. May 1990.

[2] E.D. Dichmanns and A. Zapp. Autonomous high speed road vehicle guidance by computer vision. *Proceedings of the 10th World Congress on Automatic Control.* Vol. 4. Munich, West Germany. 1987.

[3] T. Jochem, D. Pomerleau, and C. Thorpe. *Initial Results in Vision Based Road and Intersection Detection and Traversal.* CMU Technical Report CMU-RI-TR-95-21. 1995.

[4] T. Jochem, D. Pomerleau, C. Thorpe. MANIAC: A Next Generation Neurally Based Autonomous Road Follower. *Proceedings Intelligent Autonomous Systems-3.* Pittsburgh, PA. 1993.

[5] T. Jochem and S. Baluja. Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following. In *Massively Parallel Artificial Intelligence*, Kitano and Hendler (ed). AAAI Press. 1994.

[6] S.K. Kenue. Lanelok: Detection of lane boundaries and vehicle tracking using image-processing techniques. *SPIE Conference on Aerospace Sensing, Mobile Robots IV.* 1989.

[7] K. Kluge. *YARF: An Open Ended Framework for Robot Road Following.* Ph.D. dissertation, School of Computer Science, Carnegie Mellon University. 1993.

[8] K. Kluge and C. Thorpe. Intersection Detection in the YARF Road Following System. *Intelligent Autonomous Systems 3.* Pittsburgh, PA, USA. 1993.

[9] D.A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation 3:1.* Terrence Sejnowski (Ed). 1993.

[10] D.A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Ph.D. dissertation, Carnegie Mellon University. 1992.

[11] S. Rossle, V. Kruger, and Gengenbach. Real-Time Vision-Based Intersection Detection for a Driver's Warning Assistant. *Proceedings Intelligent Vehicle '93.* Tokyo, Japan. 1993.

Chapter IV: Obstacle Avoidance and Cross-Country Navigation

Introduction

The second class of behaviors addresses the problem of obstacle avoidance and, more generally, driving in cross-country environments. Those behaviors take as input a range image, construct an representation of the terrain in front of the vehicle, and evaluate a set of arcs based on this representation. The output is a vote for each arc that measures the traversability of the terrain along the arc. The behaviors differ in their internal representations of the terrain and in the way the arcs are evaluated.

Two main approaches were developed. In the first approach, the three-dimensional points from the range image are projected in a grid parallel to the ground plane. Grid cells are then evaluated for traversability using density of data points in the cell, terrain slope, and height variation within the cell. Cells that are not traversable are flagged as obstacle cells. An evaluation algorithm compares the position of the obstacle cells with the trace of each of the arcs and then generates the votes. This approach, implemented in the SMARTY system, has been demonstrated both with laser range finders and passive stereo systems. In addition to steering votes, SMARTY also generates votes for speed value and for sensor orientation; both types of votes are computed from the distribution of obstacles. Controlling the sensor orientation is especially important in the case of stereo vision in which the field of view of the cameras is fairly narrow. SMARTY generates distributions of votes several times a second. The votes are combined by the DAMN arbiter to compute a command executed by the vehicle, as previously described in Chapter II. The first part of this Chapter describes the SMARTY system in detail.

In our second approach, the points are again placed in a grid but this time the grid is used as an elevation map of the terrain around the vehicle. Data from multiple images is merged into an aggregate elevation map. At regular intervals, the behavior computes the vehicle configuration at successive points along each of the arcs. For each position, it computes a "hazard" measure which reflects, among other things, the angle of the vehicle and the undercarriage clearance. Hazard values are integrated along each arc, yielding a vote for each arc, in a way similar to the votes generated by SMARTY. This approach was implemented in the RANGER system which was also demonstrated with a laser range finder and with passive stereo systems. RANGER is described in detail in the second part of this Chapter.

Although both systems can perform real-time obstacle avoidance and cross-country navigation, SMARTY and RANGER have complementary strengths and weaknesses. SMARTY can detect small obstacles better than RANGER can. On the other hand, RANGER is better at smoothly varying terrain and negative obstacles. The complementarity stems from the fact that SMARTY uses a discrete representation of the obstacles while RANGER maintains a representation of the underlying terrain. We have experimented with combining the two approaches by having SMARTY generate discrete obstacles that are placed in the RANGER map and used in combination with the RANGER hazard measure to generate the final steering votes.

SMARTY and RANGER form the basis of the Demo II cross-country navigation system. They have been demonstrated both at CMU, as described in Chapter VI and at Lockheed Martin on the Demo II vehicles. In addition, they have been used at other sites with different sensors. SMARTY was successfully ported to the NIST platform under VxWorks using the Dornier EBK laser range finder; RANGER was demonstrated at JPL using real-time stereo vision; RANGER was used on a smaller vehicle with passive stereo vision for a planetary exploration application.

Recent additions to SMARTY include the ability to vary obstacle detection sensitivity as a function of distance to the sensor and a mechanism for eliminating spurious detected obstacles from the local map. This mechanism is implemented by attaching a detection confidence to each obstacle cell. The detection confidence increases if an obstacle is repeatedly detected over several images of a portion of the terrain. If the confidence

is too low, the obstacle is eliminated from the map. Those two additions allow for more robust driving in cluttered environments.

Both SMARTY and RANGER use an explicit model of terrain traversability. An alternative approach would be to "learn" the traversability criteria from observing actual driving through terrain, in a way similar to the approach used in ALVINN for road following. To test this approach, we developed the MAMMOTH (Modular Architecture Multi-Modal Theory Network) system which comprises of a set of neural networks which take as input video and range images and generate recommended steering arcs as output. We describe MAMMOTH in detail later in this Chapter.

MAMMOTH addresses an important issue in cross-country navigation. By using non-geometric data, e.g. a video camera, MAMMOTH takes into account the nature of the terrain, e.g., vegetation vs. dirt, in addition to its geometry. This is critical because regions that may appear as obstacles may actually be navigable once the nature of the terrain is taken into account, for example, in the case of tall grass. Conversely, regions of the terrain that are geometrically navigable may actually be hazardous for the vehicle, for example in the case of a muddy region of the terrain which would appear as a flat plane to SMARTY or RANGER. Terrain typing is therefore one of the most critical issues in achieving robust and general cross-country navigation.

In addition to the core behaviors in obstacle avoidance and cross-country navigation, we have included three more sections in this chapter on stereo vision, sonar, and teleoperation, respectively. As mentioned earlier, although our work in obstacle avoidance and cross-country navigation started by using a laser range finder, passive stereo vision is becoming an increasingly attractive option. This chapter includes a detailed description of one of our stereo systems which was used for cross-country navigation. Although this particular stereo system is not in active use in the UGV program, it was an important step in the development of the next generation stereo system, for example, the development of a frame rate stereo machine in a separate program which we intend to use on the HMMWV. Also, the algorithms described in this stereo paper are typical of the ones used in future systems.

Another sensor modality that can be used for obstacle avoidance is ultrasonic sensing. Sonars cannot be used for producing the types of terrain maps that SMARTY and RANGER need, but they are valuable for detecting discrete obstacles at relatively close range. They can be used as "soft bumpers" or as coarse mapping sensors. The GANESHA (Grid-based Approach for Navigation by Evidence Storage and Histogram Analysis) system integrates many sonar measurements into a consistent obstacle grid. GANESHA includes a filtering component that eliminates false alarms, a critical component when using sonars. In addition to obstacle mapping, GANESHA includes provisions for planning a path around obstacles and for complex operations such as docking. GANESHA was tested mostly in road environments, demonstrating the usefulness of ultrasonic sensing for outdoor vehicles. Historically, GANESHA was our first integrated obstacle avoidance system. In particular, SMARTY integrates a lot of the ideas developed for GANESHA in the algorithms it uses for managing its local map.

In many scenarios, fully autonomous cross-country navigation is undesirable either because the terrain is considered too difficult, or because human control of the vehicle is required. In those situations, a human operator must remotely control the vehicle by watching images transmitted from the vehicle. There are two main difficulties in implementing such a teleoperation system. First, the video images presented to the operator are two-dimensional, while the terrain may, in general, have significant variations in slope. Second, secure digital communications links are typically very slow and cannot handle close to video-rate transmission of images. In typical systems, transmitting one image may take several seconds even with compression.

The STRIPE (Supervised TeleRobotics using Incremental Polygonal Earth Reprojection), teleoperation behavior for uneven terrain and low-bandwidth communication addresses these problems. The key element of STRIPE is the use of the internal inertial pose of the vehicle to reproject the points selected by the operator on the actual terrain. This approach eliminates the "flat earth" assumptions commonly found in teleoperation systems. We have demonstrated STRIPE on a variety of terrain, both with simulated communication delays and by using cellular modems or packet radios for communications. STRIPE was integrated on the Demo II vehicles and demonstrated in all the intermediate UGV demos. STRIPE is described in detail at the end of this Chapter.

SMARTY: Pixel-Based Range Processing for Autonomous Driving

1. Introduction

A critical component of mobile robot systems is the ability to detect regions of the environment in which the vehicle should not travel. This is an especially important problem when a robot operates outdoors at relatively high continuous speed. In on-road driving, obstacles have to be detected at long range and very quickly to allow enough reaction time at high speeds. In cross-country driving, unsafe regions of the terrain also have to be identified at long range with the added requirement that simple strategies such as elevation thresholds do not work because of the complexity of the terrain. We will refer to regions of the environment in which the vehicle is not allowed to drive as obstacle regions or as untraversable regions without making any distinction.

The basic approach to the problem is illustrated in Figure 1: A cartesian elevation grid is built from range data and is transformed to a local map of untraversable regions which is used for generating admissible paths. Depending on the system under consideration, the implementation details of the general approach may be different. For example, the local map may be an explicit data structure [8] or it can be implicitly computed as the paths are evaluated over the grid [7][9]. Irrespective of the particular implementation, the main issue is to make the system as effective as possible in detecting obstacles as soon as they appear in the data and in using them in the path calculations as soon as possible after they are detected.

Most systems are image-based or map-based in that they have to wait until a full range image, or a full map, is processed before reporting any obstacles to the path generation module. The approach proposed here is a way to process range data pixel by pixel (Section 1.1.), updating an internal representation on the fly for each pixel (Section 3.) and correcting paths as soon as the obstacles are detected instead of waiting for a full image to be processed (Section 4.). This approach requires interleaving pixel processing, local map updating, and arc generation in a single module to eliminate the latency due to communication between modules and show timing results to validate our conclusions (Section 6.). We have implemented a driving system based on our approach to pixel-based processing and tested it in a cross-country application. The system is an extension of a previous system for off-road navigation reported in [8]. The current system uses an imaging laser range finder, described in Section 2., as the source of range data.

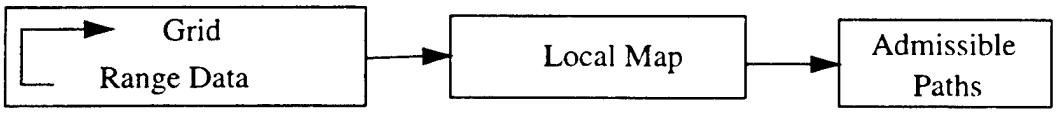


Figure 1: Perception system for autonomous navigation.

1.1. Pixel-Based Processing: Overview

The traditional approach is illustrated in Figure 1. A range image is first acquired from an active range finder or from passive stereo. Then, every range pixel in the image is transformed to local grid coordinates and stored in one cell of the grid. Finally, the grid is traversed and each cell is evaluated for traversability. Once the untraversable cells are identified, they are used for planning an admissible path. It is useful to view these obstacle cells as being maintained in a local map. This view of the traditional approach corresponds most closely to our earlier implementation described in [8] and in the stereo-based system described in [10] and [11]. We call this approach "image-based" processing.

Although this is a natural and convenient way to implement range image processing, it has many drawbacks. The first drawback is the latency in the navigation system due to the fact that the system has to wait for the processing time of a complete image before any result is produced from the perception system. The latency becomes a severe limitation with much faster acquisition rate from both passive stereo and laser range finders on the horizon.

A second problem is the fact that a system using image-based processing is typically highly sensor-dependent. In particular, it is usually tuned to a particular geometry of the imaging system. This lack of flexibility is particularly a problem when using sensors with variable fields of view in which the image structure may vary dynamically.

A more fundamental limitation of image-based processing is its dependence on a particular data format, the image array. This is appropriate for many sensors such as traditional imaging laser range finders, but it is not appropriate for other sensor modalities. For example, very fast single-line sensors have been proposed as a means to perform obstacle detection using either passive techniques [3][4], or laser ranging. In these sensors, the trace of the data is a single line. In this case, the basic unit is a single scanline which cannot be treated in the same manner as a dense 2-D image. The interest in these sensors stems from their low-cost and speed which makes them ideal for automated driving applications (e.g., [2]). Even faster laser range finders recently developed use a non standard scanning geometry in which range pixels measured on the ground may follow concentric circles around the vehicle or may follow helicoidal patterns [13]. In those cases, the concept of image, or even scanline loses its meaning.

The alternative, pixel-based range processing, is illustrated in Figure 2(b). Here, each pixel is processed individually, independently of the way it was actually acquired, through an image or through another type of scanning pattern. Every time a new range pixel is available, it is converted to a 3-D point in the coordinate system of a grid. Each cell of the grid has a current state which is updated whenever a pixel falls in the cell. Every time a cell is updated, its state is used to evaluate the traversability of the cell. The cell is placed in a separate local map if it is found to be untraversable. In the rest of the paper, we will distinguish between the *grid* which is a discrete array of cells that is updated every time a new data point is processed, and the *local map* which is another array of cells expressed in a different coordinate system. Grid cells contain a state that is used to determine traversability, while local map cells contain only a flag that indicates whether the cell is an obstacle. The distinction is made because the local map maintains information gathered over many scans but updates it at relatively low frequency, while the grid maintains information gathered over a few scans and is frequently updated.

There are many advantages in using pixel-based processing. First, it does not make any assumptions about the scanning geometry of the sensor. As a result, pixel-based processing can still be used with imaging sensors but can now be used with arbitrary scanning geometry sensors. In particular, this approach provides a simple, common framework for a number of proposed single-line sensor for fast obstacle detection. A second advantage of pixel-based processing is the ability to have better control over scheduling issues. Specifically, the generation of admissible arcs can be interleaved with the range processing by checking after each pixel is processed whether it is time to send a new steering command to the vehicle. This approach to resolving scheduling between perception and planning enables for a guaranteed command update rate.

2. Range Date Acquisition

In order to test and demonstrate our approach to pixel-based processing, he Erim laser range finder was used as the primary source of range data. This sensor is an imaging range finder that produces a 64 by 256 range image at 2Hz. The characteristics of the sensor and its used are described elsewhere [5]. Although the sensor is an imaging sensor, we are simulating a single scanner by processing each line independently and by tagging each scanline by the position of the vehicle at the time the scanline was acquired. Although all the results presented

in this paper use Erim images, the implementation is independent of the nature of the underlying data.

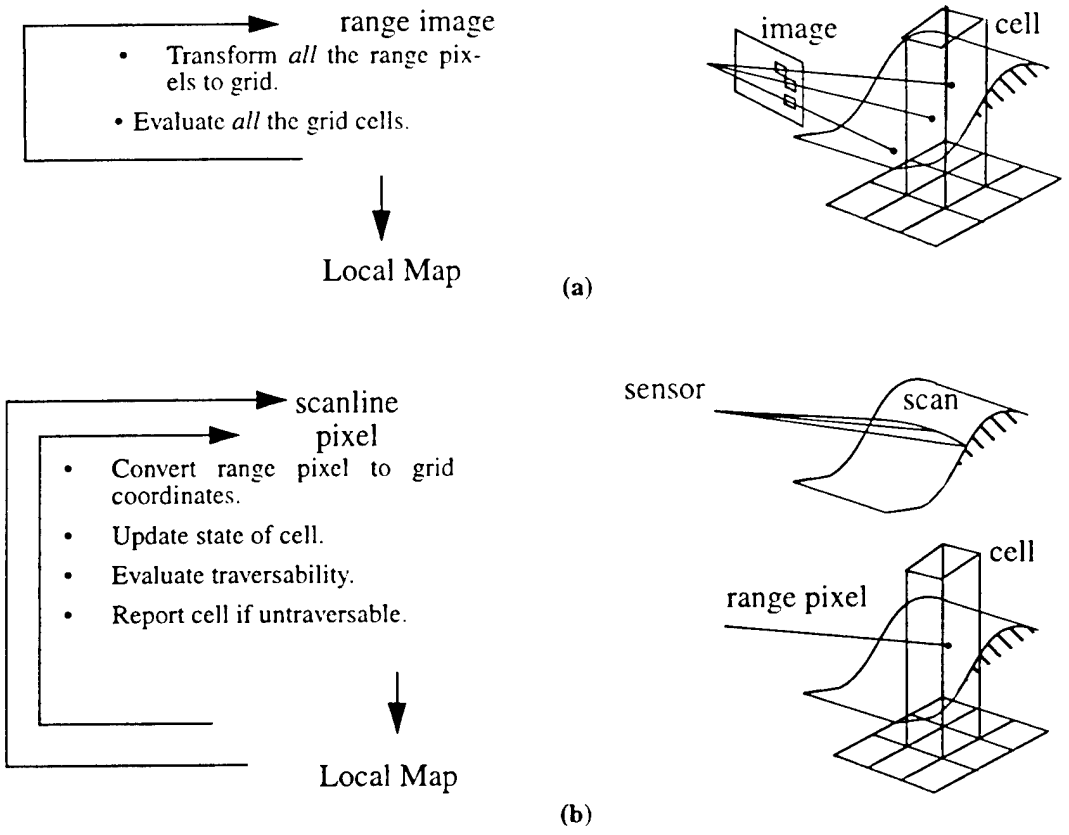


Figure 2: (a) Traditional image/grid processing; (b) Pixel-based processing.

3. Updating Cell State

The state of a grid cell is described by the following fields:

- Number of data points: N_p. N_p is the number of data points that fall in the cell.
- Average coordinate and uncertainty: $\boldsymbol{p}$, $\boldsymbol{R}$. $\boldsymbol{p}$ is the vector of average coordinates of the set of data points in the cell. $\boldsymbol{R}$ is the uncertainty matrix computed from the set of coordinates.
- Min and max elevation: z_{min}, z_{max}. z_{min} and z_{max} characterize the variation of elevation within a cell.
- Average slope and uncertainty: $\boldsymbol{a}$, $\boldsymbol{S}$; used for evaluating traversability.
- Number of scanlines: N_s; N_s is used for reducing the number of cell evaluations.
- Traversability status: T_s. T_s is true if the cell has already been evaluated for traversability and has been determined to be an obstacle cell. T_s is used for reducing the number of cell evaluations.

The average elevation and slope can be estimated recursively every time a new data point is added to a cell. If the surface is represented locally by $z = ux+vy+w$, where z is the vertical axis, then the parameter vector $\boldsymbol{a} = [u$

$v\ w]^t$ can be updated by the recursive equations: $\boldsymbol{a}_n = \boldsymbol{a}_{n-1} + \boldsymbol{K}(z_n - \boldsymbol{M}\boldsymbol{a}_{n-1})$, where $\boldsymbol{a}_{n-1}$ is the current estimate of the cell parameters, $\boldsymbol{a}_n$ is the new estimate after adding data point $\boldsymbol{x}_n = [x_n\ y_n\ z_n]^t$, $\boldsymbol{M}$ is the vector $[x_n\ y_n\ 1]$, and $\boldsymbol{K}$ is a gain computed from the uncertainty matrices of $\boldsymbol{a}_{n-1}$ and of $\boldsymbol{x}_n$. The uncertainty matrix $\boldsymbol{S}_n$ of $\boldsymbol{a}_n$ is also updated recursively by: $\boldsymbol{S}_n^{-1} = \boldsymbol{S}^{-1}_{n-1} + \boldsymbol{H}^t\boldsymbol{C}_n\boldsymbol{H}$, where $\boldsymbol{C}_n$ is the uncertainty of $\boldsymbol{x}_n$ (a 3x3 matrix), and $\boldsymbol{H}$ is the 3x3 Jacobian matrix: $[u_{n-1}\ v_{n-1}\ -1]^t[x_n\ y_n\ 1]$. The vector of parameters $\boldsymbol{a}$ is converted to a slope vector and an average elevation value in the cell. This approach has the advantage of using simple computations to update slope without having to recompute all the coordinate moments every time. Also, this gives a simple way to estimate the uncertainty on elevation and slope at each point. The estimation of $\boldsymbol{p}$ and $\boldsymbol{R}$ is conducted in a similar way. Although it may seem that this formulation would require a large amount of computation, it is very efficient in practice owing to the simple nature of the matrices involved. In fact, this approach coupled with the efficient approach to scrolling of Section 5. makes it more efficient than the traditional full image approach.

Several tests are performed every time a cell is updated to determine whether the cell is traversable or is an obstacle cell. Two tests are currently performed. The first one checks the variation of elevation in the cell according to the min, max, and average values by comparing it to a threshold. The second one uses the direction of slope. These two tests correspond to two traversability conditions introduced in [9], that is, undercarriage clearance and vehicle tilt.

Figure 3 shows an example of updating the state of a grid cell by adding pixels. Figure 3(a) is a plot of the vertical component of slope and its uncertainty as function of the number of pixels added to the cell and Figure 3(b) is a plot of mean, min, and max elevation also as function of the number of pixels. In this example, the cell was located approximately ten meters from the sensor. The pixels appear from the bottom to the top of the surface patch contained is the cell. The slope does not vary monotonically because the variation in elevation is small enough until more than ten points are added to the cell that it appears as if the terrain were flat. The cell is correctly classified as untraversable after 13 points. The example was generated from real data by plotting the state of one of the cells of Figure 4.

A cell is sent to the local map if it is found to be untraversable by computing its coordinates with respect to the local map and by marking the corresponding cell in the local map. We have used a scanline-based outer loop in the actual implementation in that the obstacle cells are sent to the local map only after the entire scanline is processed. However, as noted earlier, the notion of scanline is used only for convenience, because the data from the ranging sensors available to us is naturally organized in scanlines, and because we could tolerate the scanline delay in our navigation scenario.

Figure 4 shows an example of processing on a typical outdoor scene. The top part of the figure shows a video image of an outdoor scene and the corresponding range image.The horizontal lines overlaid on the range image show the four scanlines included in this figure, although all the scanlines are actually processed

Four instances of the grid are displayed in this figure, corresponding to the four scanlines displayed on the range image. Each display is an overhead view of the grid in which the rectangular icon indicating the position of the vehicle and numbered scale on the left indicate distance from the vehicle in meters. The footprints of the scanlines are displayed in each of the grids. The obstacle cells detected as the scanlines are processed are displayed as black dots. The grid displays include all the obstacle cells detected from the scanlines that fall inside the grid for this position of the grid reference frame. In this example, the grid is 60 meters wide and 40 meters

deep with a cell resolution of 40cm.

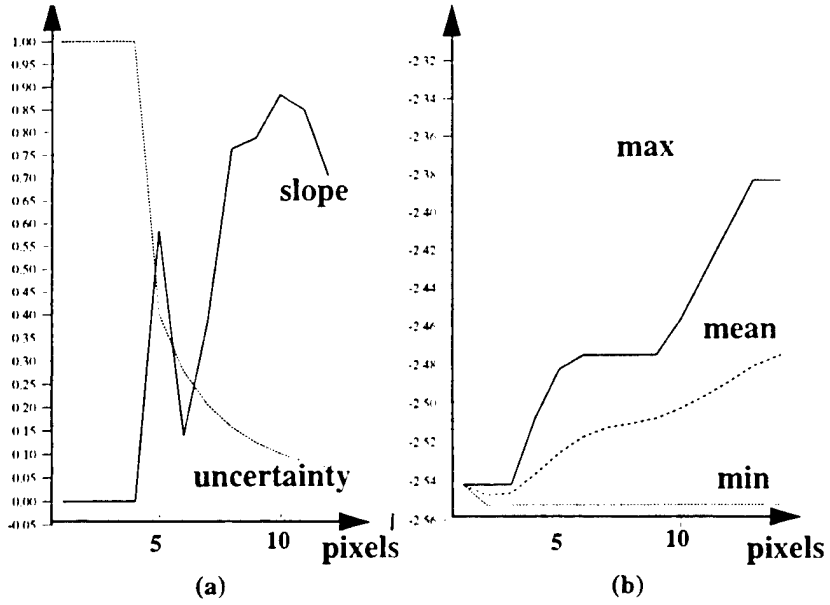


Figure 3: Updating the state of a cell; (a) slope and confidence as functions of number of pixels; (b) estimated elevation in cell as function of number of pixels.

The scanlines are processed starting with the one that is closest to the vehicle. The order in which obstacle cells are found illustrates why the pixel-based processing allows for lower latency and better system performance. The obstacles closer to the vehicle detected in line 1 are immediately reported to the local map. In general, the algorithm has the desired behavior of reporting the closest objects first. By contrast, a more traditional approach would require a relatively large map to be built before any obstacle cells are reported.

Each scanline is converted to the current grid coordinate system by using the vehicle pose at the time the line was acquired. Since the grid has a finite size, the running scanline will eventually fall outside of the grid. In t Figure 4, this happens just after line 4, at which the reference coordinate system of the grid is modified to keep the scanlines in view.

In order to avoid unnecessary cell evaluations and to avoid erroneous classifications, ithe first condition is to make sure that a cell contains sufficient information before performing the tests. A cell is evaluated only if T_s = 0, that is, it has not been classified through an earlier pass, and if $N_s > 1$. The second condition is to avoid situations in which data from a single scanline is included in a cell that lead to a numerically unstable estimation of the slope. Currently, the traversability tests are applied to a cell only if the number of points in the cell is large enough, five in the current implementation, and if the uncertainty on the parameters is low enough. The effect of using the uncertainty is to require proportionally more data points in a distant cell than in a close cell for traversability evaluation. It is a desirable behavior because a measurement at long range is less accurate than at short range and therefore requires more evidence.

4. Local Map and Arc Generation

The local map is an array of cells with a simpler structure than the grid. Local map cells contain only a binary flag indicating whether or not the cell is an obstacle, and if it is an obstacle, it also contains the coordinates of a 3-D point inside the obstacle. Storing the coordinates of the point is necessary to avoid aliasing problems when transforming the local map as we will show in Section 5.

The underlying structure of the local map is a two-dimensional array, but only the obstacle cells are used. Since the obstacle cells occupy a small fraction of the map (typically less than 10%), it is advantageous to maintain a separate list $\mathcal{L}$ of obstacle cells; each element of $\mathcal{L}$ contains a pointer to a cell location in the grid.

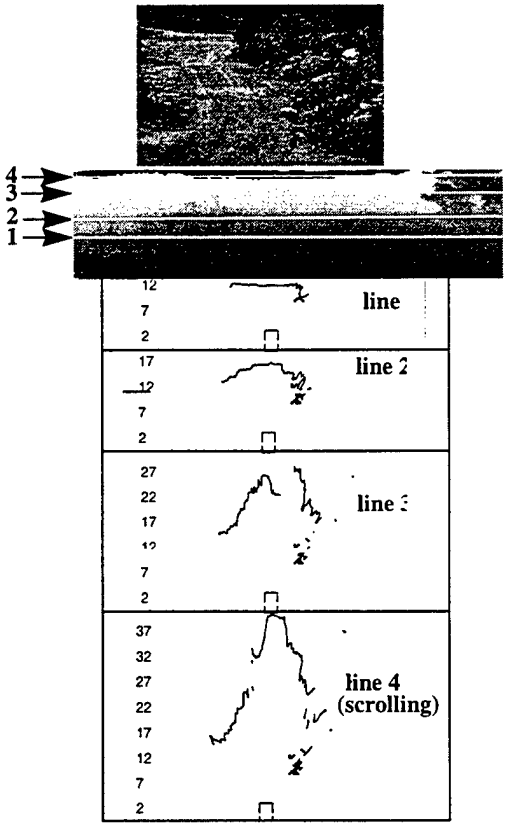


Figure 4: Scanline-based processing from a range image.

The local map is used for generating admissible steering commands. The approach used here was developed by Julio Rosenblatt [6]. We give here only a brief description of the approach and refer the reader to [6] for a detailed description of the planning architecture. The basic structure used in the arc generation is a discrete array of steering radii. A vote between -1 and +1 is associated to each arc, -1 meaning that the arc is forbidden, +1 meaning that the arc is admissible. Each steering radius corresponds to a segment of arc in the local map with its origin at the origin of the local map. Each possible arc is evaluated by computing the distance between every cell of $\mathcal{L}$ and the arc. An arc receives a vote of -1 if it intersects an obstacle cell and it receives a vote varying monotonically between -1 and +1 with the distance to the nearest obstacle cell. The function used for mapping cell distances to votes is described in [8]. A set of 39 possible arcs is used in the current implementation. The arc radius ranges from -8 m to +8 m using an approximately linear variation in curvature. After the vote for each individual arc is computed, the entire array of votes is sent to an arbiter module [6] which generates an actual steering command that is sent to the vehicle.

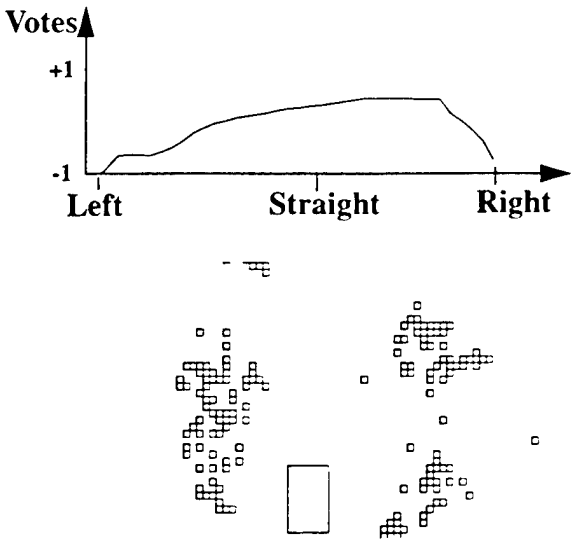


Figure 5: A local map (bottom) and the corresponding array of votes (top).

Figure 5 shows an example of arc generation from a local map. The bottom part of the figure shows the location of the obstacle cells, displayed as squares, in the local map. For reference, the vehicle is shown as a rectangular icon at the origin of the local map. In this example, the terrain is another part of the corridor shown in Figure 4 with an untraversable region on the both sides of the vehicle and with a traversable corridor curving slightly to the right. The top part of the figure shows the corresponding distribution of votes from a minimum turning radius of -8 meters to a maximum of +8 meters. The curves show the maximum votes are for moderate right turns of the vehicle and are close to -1 for left and hard right turns.

5. Maintaining Transformations and Scrolling

One of the main difficulties in this type of system is to maintain a consistent representation of pieces of data acquired at different times corresponding to different locations of the vehicle. We use vehicle pose as a tagging mechanism. Specifically, every piece of information is tagged by the pose of the vehicle with respect to a fixed reference frame at the time the information was computed. In the remainder of this section we will simply say "the pose of the data", e.g., the pose of a scanline, to mean "the vehicle pose associated with the data". We describe the various poses used in the system in the next section; we describe the scrolling used for updating the grid and the local map as vehicle pose changes in Section 5.2.

5.1. Transformations

In general, vehicle pose is the position in x, y, and z and the orientation in roll, pitch, and yaw of the vehicle with respect to a fixed reference frame. Although poses have, in general, six degrees of freedom, we limit ourselves in this paper to the three degrees of x,y position and heading because of current hardware limitations, although the algorithm is not substantially different. The accuracy of the dead-reckoning system is on the order of 1% of distance traveled [12], which is sufficient for our application.

Figure 6 lists the five types of poses that are used by the system. The first column of the table lists the symbols used for the poses in the discussion below; the second and third columns give a summary description of the poses and the time at which they are acquired; the fourth column indicates which data type the poses are

attached to.

Pose	Description	Updated When	Data
V_l	Pose at the time the scanline is acquired.	New scanline is being processed.	Scanline
V_b	Pose at the beginning of the image.	New image is read.	Image
V_e	Pose at the end of the image.	End of image is reached.	Image
V_g	Pose in which the grid is expressed.	Scanline is outside of the grid.	Grid
V_m	Pose in which the local map is expressed.	Arcs are sent to the planner.	Local Map

Figure 6: Transformations and their interrelations.

The first pose V_l is computed every time a new scanline is read from the sensor and is attached to the scanline. V_l is read directly from the controller in the case of a one-line sensor; it is interpolated between the beginning and the end of the image in the case of an imaging sensor as described below. As noted earlier, the concept of scanline is used for convenience only and can be replaced by V_r, the vehicle pose for a single range pixel, for an arbitrary scanning geometry. Such pose data is available from scanners such as [13].

The next two poses, V_b and V_e, are meaningful only when an imaging sensor is used. They are used for correcting the smearing of the image due to vehicle motion by interpolating pose of any scanline in the image between V_b and V_e. Currently, the interpolation is a simple linear interpolation which performs well for moderate speeds and rates of turn.

The next pose, V_g, indicates the origin of the grid in which the coordinate conversion and obstacle classification are taking place. Whenever a new scanline of pose V_l is processed, its data points are converted from local sensor coordinate system to a grid coordinate system by $V_g.V_l^{-1}$. Unlike the previous poses, V_g is not read from the controller but is computed from the other poses in order to ensure that the grid covers an area large enough in front of the vehicle. This is the object of the *grid scrolling*. The first time data are processed, V_g is set to the pose of the first scanline V_l^0, the origin of the grid is placed at V_g and all the computations are done with respect to this origin. As the vehicle travels, it reaches a point a which a new scanline, of pose V_l^1 is partially outside of the bounds of the grid. When this condition occurs, V_g is set to V_l^1 and all the cells in the grid are transformed to this new reference frame. This procedure is repeated whenever a new scanline falls outside of the grid.

The last pose, V_m, is the pose with respect to which the elements of the local map are expressed. A new obstacle cell is added to the map by transforming its coordinates in the grid to the local map coordinates using the transformation $V_m.V_g^{-1}$. A new V_m is read from the controller every time the system determines that it is time to send new arcs to the planning system. When this condition occurs, the system performs a local map scrolling operation in which all the elements are converted from their coordinates in the previous V_m to the new V_m.

5.2. Scrolling Algorithms

A scrolling grid or a local map involves applying the appropriate transformations to the cells and re-arranging the cells according to the new coordinates. The implementation of both grid and local map require a efficient scrolling algorithm, for the scrolling is potentially expensive and may introduce undesirable delays. A straightforward implementation would be to initialize a new array, to transform all the cells of the old array into the new one, and to copy the new array into the old location. This simple algorithm is potentially expensive because it requires initializing and traversing a potentially large array. This algorithm is inefficient because the

array is typically very sparse, especially in the case of the local map in which less than 10% of the cells are used at any given time. From the point of view of navigation, this algorithm has the undesirable side effect that it takes the same amount of time irrespective of the number of active cells in the array.

A better approach is to maintain a marker M for each cell. The marker is used to indicate whether the cell has been visited since the last time the array was scrolled. In addition, a list $\mathcal{L}$ of active cells is maintained. In the case of the grid, the active cells are those cells to which more than one data point contributes from the current grid pose. In the case of the local map, the active cells are the current obstacle cells. With these definitions, the scrolling proceeds as follows: First, the current marker value, M_0, is incremented, then the coordinates of each cell pointed to by the elements of $\mathcal{L}$ are computed, the corresponding pointer is updated and the marker of the cell is set to the new value M_0. Two cases may occur after the transformed cell C' of a cell C of $\mathcal{L}$ is computed. If the marker of $M(C')$ is different than the current value M_0, then C' should be considered as a new cell and should be initialized, otherwise, its current value is updated by adding the data from C'. The way the update is performed depends on the array being scrolled. In the case of the local map, no computation is needed because C' is already listed as an obstacle and does not need to be updated further. In the case of the grid, the data in C', such as mean elevation, slope, etc. is updated by merging the data from C. The updates of slope and mean elevation are performed using the algorithm of Section 1.1.. After all the cells in $\mathcal{L}$ have been updated, $\mathcal{L}$ is traversed one more time to delete entries C such that $M(C) \neq M_0$. Such entries are cell locations that are not used in the scrolled array.

The use of markers is described in the context of scrolling but the same mechanism is used for adding new cells to the grid or to the local map. Let C be the cell in which the new data are to be added, if $M(C)$ is different from M_0, then C is initialized, the new data is copied in C, and a pointer to C is added to $\mathcal{L}$; otherwise, C is updated with the new data. This algorithm allows for efficient scrolling of the grid and local map arrays by initializing and updating only the cells that are needed. In particular, it has the desired behavior of being faster when no active cells are present in the array.

6. Performance

In the current implementation, the driving system is implemented as shown in Figure 2: The main loop acquires and processes one scanline at a time, updating grid cells as described in Section 3. and scrolling the grid when necessary as described in Section 5. In addition, the scrolling local map and arc generation of sections 4. and 5. are activated at regular intervals. The current interval is 100ms, although the actual frequency of map scrolling and arc generation must be computed based on the desired speed of the vehicle. The time performance of the system is shown in Figure 7.

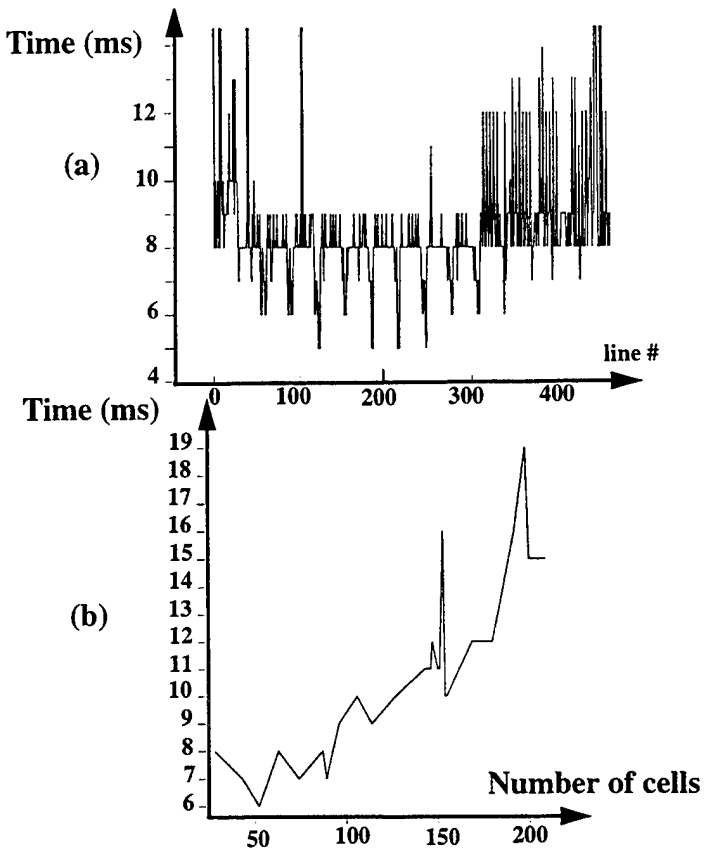


Figure 7: (a) Grid processing time per scanline; (b) local map processing time as function of the number of obstacle cells.

Figure 7(a) is a plot of computation time per scanline. The vertical axis is the time in milliseconds spent processing each scanline and the horizontal axis is the scanline number. There are on average 256 points per scanline. The graph shows that the time per scanline is 8ms on average with the periodic drops being due to the fact that not all pixels of a scanline are processed at far range. Figure 7(b) is a plot of the computation time used for scrolling the map and generating the arcs as a function of the number of obstacle cells. Although the absolute time values are clearly dependent on the platform used, in this case a Sparc II station, the graphs show that our proposed architecture for scanline- and pixel-based processing allows high performance and low latency using conventional hardware. In particular, an obstacle is taken into account in the arc calculation at most 100ms after it is detected with very little additional delay (19ms is the worst case in this example). The data was collected on the course used in [8].

7. Conclusion

An efficient approach to range data processing for autonomous driving was developed. The approach uses the concept of pixel-based processing in order to eliminate latency in the system and to allow for efficient merging of range data processing, obstacle detection, and arc generation. We have demonstrated the system using the Erim imaging laser range finder to simulate a single scanline sensor.

This work improves on our earlier system for autonomous navigation [8] by eliminating the delays in image processing and by eliminating variable communication delays through tighter integration with predictable cycle time. Our current work concentrates on demonstrating the system on long-distance missions such as in

[8]. Also, we will demonstrate the flexibility of the system by using it with other sensors, such a fast single scanline sensor and passive stereo. Finally, we are planning on porting the system to a real-time operating system in order to eliminate the remaining spikes in computation time profile due to resource contention between processes.

Bibliography

[1] O. Amidi. *Integrated Mobile Robot Control.* CMU-RI-TR-90-17, The Robotics Institute, Carnegie Mellon, 1990.

[2] J.L. Bruyelle, J.G. Postaire. Direct Range Measurement by Linear Stereovision for Real-Time Obstacle Detection in Road Traffic. Proceedings IAS-3. Pittsburgh. 1993.

[3] S.X. Godber, M. Robinson. Three-Dimensional Machine Vision Using Line-Scan Sensors. *Proceedings SPIE Vol. 1823.* 1992.

[4] S.X. Godber, M. Robinson, P. Evans. Stereoscopic Line-Scan Systems for Robotics Control. *Proceedings IAS-3.* Pittsburgh. 1993.

[5] M. Hebert, E. Krotkov. 3D Measurements from Imaging Laser Radars. *Image and Vision Computing 10(3).* 1992.

[6] Keirsey, D.M., Payton, D.W. and Rosenblatt, J.K. Autonomous Navigation in Cross-Country Terrain. In *Proceedings Image Understanding Workshop.* Cambridge, MA. 1988.

[7] A. Kelly, T. Stentz, M. Hebert. Terrain Map Building for Fast Navigat ion on Rugged Outdoor Terrain. In *Proceedings of the SPIE Conference on Mobile Robots*, 1992.

[8] D. Langer, J.K. Rosenblatt, M. Hebert. An Integrated System for Off-Road Navigation. *Proceedings IEEE International Conference on Robotics and Automation.* San Diego. 1994.

[9] K. Olin, D.Y. Tseng. Autonomous Cross-Country Navigation. *IEEE Expert*, 6(4). 1991.

[10] L. H. Matthies. Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time Implementation. *IJCV, 8:1.* 1992.

[11] L. H. Matthies, P. Grandjean. *Stochastic Performance Modeling and Evaluation of Obstacle Detectability With Imaging Range Sensors.* Technical Report JPL-93-11. JPL. 1993.

[12] C. Thorpe, O. Amidi, J. Gowdy, M. Hebert, D. Pomerleau,. Integrating Position Measurement and Image Understanding for Autonomous Vehicle Navigation. *Proceedings Workshop on High Precision Navigation.* Springer-Verlag Publisher. 1991.

[13] H. Verdun, G. Stevenson. 1.54 μm Solid State Laser Radar for Obstacle Avoidance. Technical Report. Fibertek Inc. 1992.

RANGER: A Feedforward Control Approach to Autonomous Navigation

1. Introduction

RANGER is an acronym for Real-time Autonomous Navigator with a Geometric Engine. This report describes the "geometric engine" part of RANGER, the state space model, and the associated control algorithms.

A new approach to the high-speed autonomy problem is presented; the approach is fundamentally based on the state space representation of a multi-input multi-output linear system and the problem analysis presented in [28]. The system closes the overall perceive-think-act loop for a robot vehicle at a relatively high update bandwidth and incorporates both somatic and environmental feedback.

A high fidelity feedforward actuator dynamics and terrain-following model is introduced; the model formulates the local navigation problem as an optimal control problem. Feedforward solves the historically significant clothoid generation problem. C-space combinatorics explosion is solved by planning in actuation space. The system feeds steering commands to the state space model, and it uses the terrain map to generate the near clothoid response of the vehicle naturally and directly with none of the algorithmic sensitivities of classical solutions. Obstacle avoidance, path planning, path generation, and path following algorithms, which are based on the state space model, are introduced. These algorithms are stable and reliable at 10 mph on rough terrain and promise to maintain stability at 20 mph and above.

The system is concerned with the high level coordinated control problem. It is not concerned with the specific control of actuators themselves. Further, while it can accept a specification of a strategic goal, such as a path to follow, or a direction to prefer, it cannot generate its own strategic goals. Therefore, it solves the *local* planning problem for autonomous vehicles. The local planning problem includes obstacle avoidance and goal seeking.

The system can be classified as an *intelligent, predictive controller* for autonomous vehicles. It is intelligent because it uses range images that are generated from either a laser rangefinder or a stereo triangulation rangefinder in order to perceive the immediate environment of the vehicle. It is predictive because it reasons based on its knowledge of its own capability to react to hazards in real time as well as the transfer function of the vehicle which relates motion commands to the response to those commands. This transfer function is concerned with the geometry, or *kinetics*, of vehicle motion. There is no explicit representation of force in the system, so it is basically a geometric one.

The system is a state space controller because it explicitly forms an expression of the vehicle dynamic state vector in order to predict the hazard signals upon which decisions are based. The process, which converts terrain shape and command inputs into the associated response of the vehicle, is a *constrained multidimensional differential equation.*

Although the word 'reasons' is used above, the system takes a traditional control systems view rather than an artificial intelligence view of the autonomous navigation problem. The path planning problem is regarded as trivial due to the nonholonomic constraint of Ackerman steering. The reason for this is that the region in space that the vehicle can physically reach which is also within the usable field of view of a sensor decreases very rapidly in size as the speed increases.

As of this writing, the system has achieved 15 km excursions, average speeds of 15 km/hr and intermittent speeds of 20 Km/hr and all of these achievements are unprecedented. It owes its success to an extensive analysis of the problem it is intended to solve.

1.1. High-Speed Autonomy

One of the unavoidable realities of safe, high-speed autonomy is that any vehicle requires significant time and space to react to external events. With vehicle and computational latencies largely fixed, the only practical avenue available is to actively search for hazards at increasing distance from the vehicle. This implies increased demands on sensor resolution and overall system accuracy, and increased sensitivity of planning decisions to small model and sensory errors because the system cannot wait until geometry is more favorable in its assessment of hazards. A real-time autonomous navigator actively manages its field of regard instead of its reaction time because the latter cannot be feasibly altered enough to make a difference.

Unavoidable system latencies and limited maneuverability take on a new level of significance. A high-speed vehicle is committed to travelling a certain trajectory for a significant distance before any decisions made in the planner can be enacted in hardware. Further, dynamics and actuator response limit the ability of the vehicle to alter its trajectory significantly after the actuator commands are received. These hard physical limits on braking and steering imply that there is no point in wasting computational bandwidth assessing hazards in regions where the vehicle is either already committed to go or where it physically cannot go. The size of the region that the system truly has an option of traversing rapidly decreases in size as speeds increase. This of course implies a reduced ability to avoid obstacles, but that is a fundamental physical cost of higher speeds.

Planning algorithms that directly incorporate knowledge of such dynamics enjoy significant performance improvements. A control systems view of a high-speed vehicle leads to complete elimination of the historically significant problems of C-space combinatorics explosion and nonholonomically constrained path generation. The problem remains one of search, but physical dynamics amounts to an overwhelming constraint. Of all possible trajectories in C-space, only a relative handful are both dynamically feasible and spatially distinct enough to warrant consideration.

1.2. Commentary

The traditional hierarchical view of the problem arises perhaps from the axioms of structured programming from software engineering and the artificial intelligence view of hierarchical planning. In this view, planners generate command inputs to controllers which do their best to follow them. While these are excellent tools, there is a point at which hierarchy becomes counterproductive when it is mapped onto contemporary computers and contemporary vehicles. In a real time context, response time and throughput requirements are absolute metrics of success and tightly coupled embedded systems are the traditional practical solution to such problems.

A basic assumption of the hierarchical view is that controllers are empowered to execute commands given to them. For the problem solved here, this is not the case. Indeed, there is very little choice available to a vehicle controller about what the vehicle will do over the short term, and any planner needs to incorporate this knowledge into its model of the vehicle-environment interaction.

When the point is reached when planners must incorporate explicit high fidelity feedforward models of vehicle dynamics and system latencies, the traditional hierarchy has, in fact, been broken because the planner has become a feedforward controller. The question becomes not *what will I tell* the vehicle to *do*, but rather *how will it respond* to this *request*. An HMMWV is a massive system and its actuators can only generate propulsive and steering forces which are a very small fraction of the total inertial force on the vehicle. Hence, the vehicle goes where it wills and a hierarchical planner-controller approach to the problem of controlling an HMMWV at high-speed is an indirect attempt to defy physics.

The only model of a high-speed conventional vehicle of practical validity in autonomy is a coupled, constrained, nonlinear, multidimensional differential equation. Idealized kinematic models of vehicle steering are a major cause of the brittleness of kinematic arc-based planners, which has not been generally recognized.

1.3. Differences from Important Historical Systems

RANGER differs from all historical precedents known to the author in two ways. First, the system explicitly evaluates vehicle safety in terms of timing, speed, resolution, and accuracy in real-time and employs many adaptive mechanisms which optimize performance. In particular, explicit computation of stopping distance and required minimum throughput is performed and these considerations and others form the basis of an elegant adaptive perception mechanism which makes unprecedented vehicle speeds possible. Second, the system uses relatively high-fidelity feedforward models of system dynamics and terrain following and is configured as a tightly coupled controller.

This section summarizes the capabilities of some of the most significant historical off road systems. The list is not exhaustive, nor is it a complete discussion of each system provided. Unfortunately, little has been published to date on some systems, and available publications do not address some of the issues investigated here.

1.3.1. Hughes ALV

The Hughes Autonomous Land Vehicle [9] was one of the first off-road mobile robots ever developed. The system generalized the notion of an obstacle to include any vehicle configuration that is unsafe. This allowed navigation over terrain of any surface shape. Based on the ERIM rangefinder and a highly terrainable vehicle, it achieved speeds up to 3.5 km/hr, over distances exceeding 500 meters, and running times averaging 20 minutes.

The speed of this vehicle was limited by many factors including the complexity of the perception and planning processing, and the speed of communication with off board processing. Robustness was limited by low fidelity modeling of steering constraints, poor terrainability and maneuverability of the vehicle on muddy soil, inability to detect small, but dangerous obstacles (steel fence spikes), constrained excursion due to line-of-sight radio communications, local planning minima, and bugs in the high-level planner.

The system discussed here differs from the ALV in its real-time minimalist approach to the problem. The adaptive perception algorithms are considerably faster than the Connection Machine implementation of the ALV algorithms. It also discards the costly energy minimization interpolation scheme of the ALV in favor of a temporal interpolation of the hazard vector signal. The system also differs in its high-speed design incorporating a differential equation state space model.

1.3.2. JPL's Robby

The JPL rover [46] was the first system to drive autonomously across outdoor terrain using dense stereo data, rather than sonar or laser rangefinder data. This system was demonstrated in a 100-meter cross-country demo in Sept. 1990. It has achieved an average speed of 100 meters in 4 hours (7.0 mm/sec) in a start/stop mode. The total elapsed time in this case is not very meaningful, since it included downtime while the system was being debugged and numerous stops to evaluate performance[1]. The vehicle speed was mechanically limited to 4 cm/sec and off-board communication throughput was limited. Cycle times were on the order of 30 secs for planning and 10 secs for perception. Passive stereo was used in the perception system. Later runs were able to achieve speeds of 4 cm/sec.

The system discussed here differs from Robby in that it attempts both wide excursion and high-speed. It is difficult to assess the degree to which computer evolution alone has made this possible.

1. Larry Mathies, personal communication.

1.3.3. JPL's Rocky Series

The Rocky series of Microrovers [43] are small scale prototypes intended as testbeds for simple behavior-based control systems. They are intended to produce a flight rover for the MESUR pathfinder mission of 1996. These robots are tested in the context of operations in the vicinity of a planetary lander, so large traverse distance or high-speed are not being pursued.

The system discussed here differs from the Rocky series in its deliberative approach to autonomy and its attempt to achieve wide excursions at higher speeds. It borrows the minimalist configuration principle in an attempt to optimize real-time performance.

1.3.4. CMU's FastNav

The FastNav project concentrated on achieving high vehicle speeds and robust obstacle detection on fairly benign terrain. The Cyclone single scanline laser rangefinder was used as the basis of the perception subsystem. It was used to check a global path for discrete obstacles, such as trees, rocks, and ditches, and stop accordingly. Obstacle detection was based on detecting deviations from a flat world. Path tracking used a position estimation system based on inertial navigation, dead reckoning, and the GPS satellite navigation system.

FastNav [41] made use of simple tire-soil interaction and vehicle dynamics models to navigate on locally-flat terrain with sparse obstacles. Path tracking was demonstrated at 30 km/hr and obstacle detection was demonstrated at 3 m/s.

Some of the reliability problems that were demonstrated include high sweep rates of the Cyclone scanner due to vehicle pitching, and poor accuracy of the inertial navigation system. Early problems with instability of the tracking algorithm were fixed by incorporating a first-order model of the steering dynamics into a feedforward compensator. The longest autonomous run was 1 kilometer[1].

RANGER borrows heavily from the FASTNAV project in three ways. First, steering dynamics feedforward is generalized to a complete state space rough terrain model. Second, the mere fact that such speeds were possible pointed directly to a problem with the way in which nonadaptive image processing was a major cause of poor real-time performance. The line scanner has been an important existence proof that the bare throughput requirement at these speeds was far less severe than it appears to be. Third, the pure pursuit path tracker is an adaptation of the FASTNAV tracker for rough terrain.

RANGER also differs from FASTNAV in that it is a cross-country system, not a dirt road system. Full 3-D models are employed and the considerable difficulties of rough terrain drive many aspects of the design.

1.4. Requirements

The design of the system is based on these top level requirements. They are the highest level design drivers that influence the design to a large extent. Certain of the components would be unjustified for systems with other requirements, such as slow, start-stop, or omnidirectional vehicles.

Rough Terrain: Rugged, sparsely vegetated, mostly dry terrain is the target environment within which the vehicle must navigate safely. A flat world assumption, along with discrete obstacle detection is not considered valid in the target terrain.

Continuous, high-speed: Speeds which approach the speeds at which humans can drive the same terrain (for

1. Much of this is personal communication with Sanjiv Singh and Jay West. Existing documentation does not cover the issues mentioned completely.

example, 20 mph) are targetted.

Ackerman Steer Vehicle: While the Ackerman steering configuration is not ideal for planning purposes, most large vehicles intended for human use employ this configuration, so the restriction to this class of vehicles is not as limiting as it may appear. In any case, the maneuverability of such vehicles is dramatically different from other alternatives, particularly at speed, and the report will show that efficiency demands that a special case solution for this class of vehicles be adopted. Nevertheless momentum and the nonholonomic constraint limits maneuverability of all high-speed vehicles, so many of the results here apply generally.

Robustness: The system is intended for relatively long traverses of suitable terrain. While robustness is difficult to quantify, it is the intention that reliability be sufficient for routine autonomous excursionsfor many kilometers over suitably chosen terrain.

2. Analytical Basis of the Concept

RANGER is an attempt to design an optimal system from a systems perspective. A consistent requirements analysis was conducted that identifies the interrelationships of requirements and that attempts to optimize the system performance as a whole. Consistent resolution and accuracy goals are imposed on the design so that all subsystems are equally well and equally poorly able to do their respective jobs. In many cases, *trade-offs are managed in real time in the code itself.*

This approach arose for historical reasons. Earlier versions of the system had no basis for allocating resources to subproblems so all were optimized at their own level. This led to "near perfect" perception and planning algorithms which did an excellent job of predicting collisions with hazards that they were too busy *thinking* about to *do* anything about.

The system eliminates the lower levels of control, which traditionally attempt to follow a path provided by a higher level agent because the problem is too complex to solve by hierarchy. The system works in actuation space and incorporates feedforward. For this reason, it is effectively a *controller* and not a planner.

The system that emerges from these considerations is unique. It is a real time control system that operates on environmental and navigation sensory inputs and incorporates a high-fidelity model of both the plant and the environment. It is tightly coupled and it merges the traditional hierarchies into one conceptual layer that closes the perceive-think-act loop at high update bandwidth. The system is based upon a large body of analyses presented in [28]. A brief discussion of the most significant aspects of the design is presented below.

Guaranteed Safety: The system directly implements the policy of guaranteed safety by reasoning in real time about the four dimensions of safety (i.e., timing, speed, resolution, and accuracy) and by adapting its perception and planning subsystems to comply directly with the need for safety.

Adaptive Regard: Adaptive Regard is a mechanism for ensuring that the system minimizes the spatial extent of the region it perceives based on vehicle maneuverability so that speed is maximized without compromising safety. The principle of adaptive regard is to scan for hazards only in the obstacle detection zone. This is the region of space that the vehicle still has an option of traversing. The system implements adaptive regard by adapting its obstacle avoidance behavior to guarantee that an impulse turn can always be executed should an obstacle be detected.

Fidelity Adaptive Planning: This principle surrounds hazards by a small buffer region in actuation space in order to compensate for any inaccuracies in the computations. It is implemented here as *Gaussian filtering* of the steering vote vector.

Feedforward Path Generation: The configuration space of the high-speed nonholonomic vehicle is degenerate at high-speed and the attendant clothoid generation problem is impossible to solve in practical terms. The

"path generator" for the system generates, not candidate paths, but candidate command signals *expressed in actuation space* (in terms of curvature and speed as a function of time), and uses a high fidelity simulator to solve the differential equations of motion in order to determine the exact response to these candidate commands. This strategy has the following advantages:

- The paths generated *meet the mobility constraints of the vehicle by construction*, so the difficult and often impossible problem of conversion from C space to A space is completely avoided. Instead, the reverse process of dead reckoning is used in the simulator. This is a kind of feedforward.

- The paths coarsely span the entire region of space that the vehicle can reach, so no real alternatives are missed.

Feedforward State Space Controller: The system represents hazards as sampled time signals in a kind of multidimensional hazard space. The essential obstacle avoidance mechanism is the *minimization of an optimal control functional* in this space. Interpolation is performed in time along the vehicle state vector rather than in the space of the terrain map.

Continuity Assumption: The assumption that important aspects of vehicle maneuverability and system computational performance will not change significantly from one cycle to the next is called the *continuity assumption*. This assumption is used to resolve some of circular issues related to the coupling between adaptive perception and adaptive regard.

Terrain Smoothness Assumption: The terrain smoothness assumption is required in order to implement a terrain map representation of the environment. It is also used to justify the interpolation of small unknown regions.

Unknown Hazard Assumption: The unknown hazard assumption involves the belief that large unknown regions in a terrain map are unnavigable by default. This key assumption permits the system to operate robustly near large holes or near the edge of a cliff.

3. Hierarchical Planning and Arbitration

The *local minimum problem* is perhaps the most serious threat to robustness of any local obstacle avoidance strategy for autonomous navigation. In order to achieve robust navigation, a sound strategic plan that avoids these minima is a solution that works well in practice. In order to enhance robustness, the system supports the input and continuous monitoring of strategic goals.

The system defines a *strategic goal* which instantaneously may be any one of the following:

- follow a predefined path

- drive toward a predefined point

- maintain a fixed compass direction

- maintain a fixed curvature

In addition to the strategic goal, a *tactical goal* is to simultaneously avoid all hazardous conditions including:

- tipover

- body collision (high centering)

- wheel collision
- entering unknown terrain

Extensions to other hazards are simple to perform.

In situations where a hazard is detected that lies in the way of achieving the strategic goal, the tactical goal will override the strategic one as long as any hazard exists. The output of the steering arbiter may be very discontinuous if a hazard suddenly appears or disappears and because high steering rates constitute a hazard by themselves, the steering arbiter output is smoothed to remove steering discontinuities except when drastic action is needed. That is, the strategic steering output is smoothed, whereas the tactical one is not.

The arbitration scheme might be called a *prioritized sort*. The principles of this technique are:

- Obstacle avoidance always overrides tracking if it chooses.
- Tracking chooses the best path, which obstacle avoidance allows when it is allowed control.

This approach has the side effect that following of an extended feature will emerge naturally, and the system will immediately take the opportunity to reacquire the strategic goal if an opportunity presents itself. While this describes the emergent behavior, the algorithm itself is to:

- sort the votes of the tactical planner using the corresponding strategic votes as the key
- choose the first path in the sorted list which satisfies obstacle avoidance

Effectively, if obstacle avoidance does not care which direction to choose, the strategic planner is given control. If it does care, it will continually veto the strategic votes in order of strategic preference until a safe one is presented to it. If no safe path exists, a stop command is issued.

4. Path Tracking

The only nontrivial aspect of tracking a strategic goal is high-fidelity tracking of a convoluted path over rough terrain.

4.1. Adaptive Pure Pursuit

The strategic controller incorporates an adaptive path tracker that is based on the *pure pursuit* algorithm. The pure pursuit algorithm has been around for some time [41]. It is basically a proportional controller formed on the heading error computed from the current heading and the heading derived from the current vehicle position and a goal point on the path. The goal point is computed by finding the point on the path that is a predetermined distance from the current vehicle position (Figure 1).

There are many variations possible on the basic algorithm. Heading is measured at the center of the rear axle. The proportional gain is normalized by the lookahead distance L. This can be viewed an adaptive element or, more simply, as a unit conversion from heading error to curvature. Indeed, the ratio ψ_{err}/L is the average curvature required to reacquire the path at the goal point.

A limiter is used to ensure that the curvature is limited at higher speeds to prevent rollover. Another limiter maintains the angular acceleration below a threshold. These measures ensure vehicle safety and directly prevent instability at the cost of an inability to track high curvature paths at a high speed. However, vehicle dynamics prevent tracking high curvature paths at a high speed anyway.

Large tracking errors or, equivalently, too short a lookahead distance or too high a gain all result in servo instability. This is a well-known problem with pure pursuit which can be addressed with feedforward.

A few modifications are introduced to adapt pure pursuit for rough terrain. First, extremely large tracking errors must be acceptable to the servo without causing instability. This is managed by two devices indicated in Figure 2.

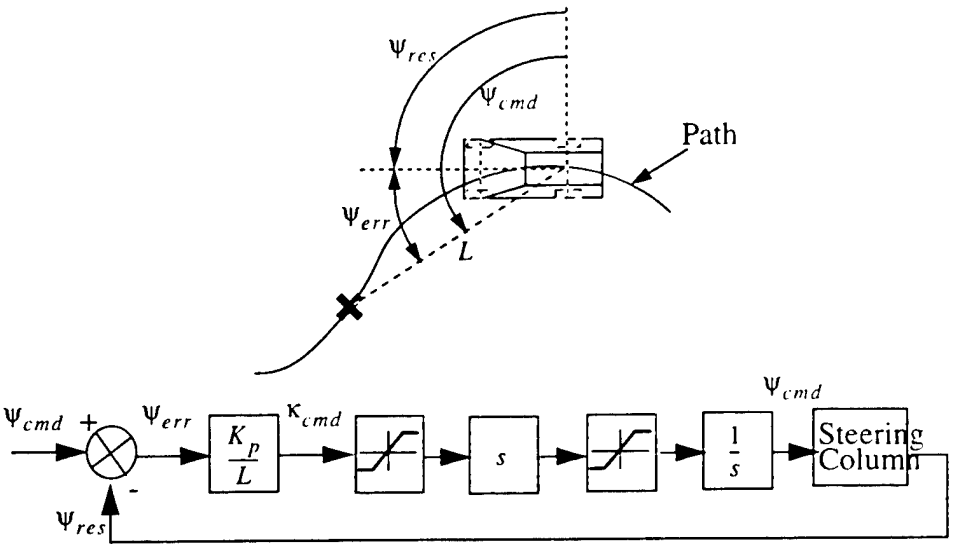
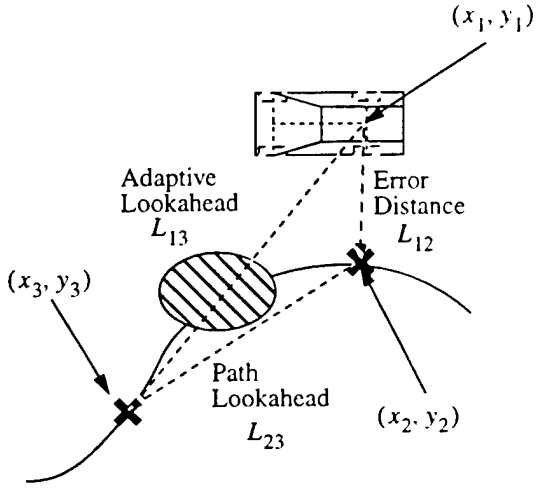


Figure 1: Basic pure pursuit algorithm.




```
/*
** Adaptive Pure Pursuit
*/
min = HUGE; way_pt = close;
while(L23 < L12 + lookahead)
  {
  x3 = path_x[way-pt];
  y3 = path_y[way_pt];
  if( L13 < min )
    {
    close = way_pt;
    }
  way_pt++;
  }
way_pt = close;
while(L13 < L12 + lookahead)
  {
  x3 = path_x[way-pt];
  y3 = path_y[way_pt];
  goal_pt = way_pt;
  }
```

Figure 2: Adaptive pure pursuit.

Instead of using the current vehicle position as the origin of the lookahead vector, the system maintains a running estimate of the point on the path which is closest to the current vehicle position. This is done because it is very expensive to search the entire path each iteration. In doing so, the system is assuming that the vehicle will never have a heading error which exceeds 90 degrees for an extended period of time. This is a *monotone arc length assumption.* This assumption completely eliminates the overwhelming computational cost of simpler implementations of the algorithm.

The lookahead distance is adaptive to the current tracking error, increasing as the error increases as indicated in the accompanying code fragment. The first while loop is responsible for maintaining a running record of the close point, point 2. It searches through an arc length window which adapts to the path tracking error. As the error gets larger, this loop will cause the close point to jump over high curvature kinks in the path as they become less relevant at the resolution of the tracking error.

The second while loop computes the goal point in an identical manner. It basically moves point 3 forward until it falls outside a circle centered at the vehicle whose radius is the sum of the error distance and the nonadaptive lookahead. In this way, when an obstacle avoidance maneuver causes significant path error, the algorithm will search to reacquire the path on the other side of the obstacle instead of causing a return to the front of the obstacle.

Notice that under normal circumstances when the vehicle is tracking the path, the close point is at the vehicle position, the error distance is zero, and the adaptive lookahead is the nonadaptive lookahead. In plain terms, the algorithm gracefully degenerates to classical pure pursuit when obstacle avoidance is not necessary.

4.2. Feedforward Pure Pursuit

An upcoming section will present a high fidelity feedforward simulator that models many aspects of vehicle kinetics. A feedforward option in the tracking algorithm incorporates the output of this simulator into the tracker. The cost of a feedforward simulator must be borne, so an additional feedforward element in the tracker is available for free. The basic idea is as follows. First, at each point in the simulation, evaluate the distance from the vehicle to the goal point. Second, the candidate command that comes closest to the goal point becomes the vote of the strategic controller. Such an algorithm provides excellent path following in three dimensions over rough terrain.

The concept is indicated in the following figure. During a high curvature turn at speed, the feedforward simulator skews the response curves toward the current steering direction.

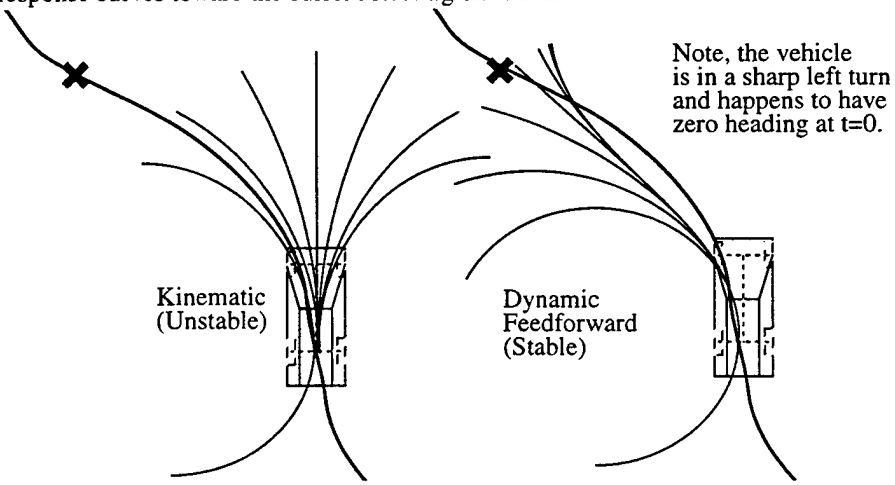


Figure 3: Feedforward pure pursuit.

A kinematic tracker would issue a hard right command in the situation depicted above, whereas a dynamic one would recognize that such a command would actually *increase* the tracking error. A dynamic tracker would issue a zero curvature command and would correctly acquire the goal point as the steering wheel slowly turned toward zero. The following figure illustrates the performance of the feedforward algorithm relative to the kinematic one.

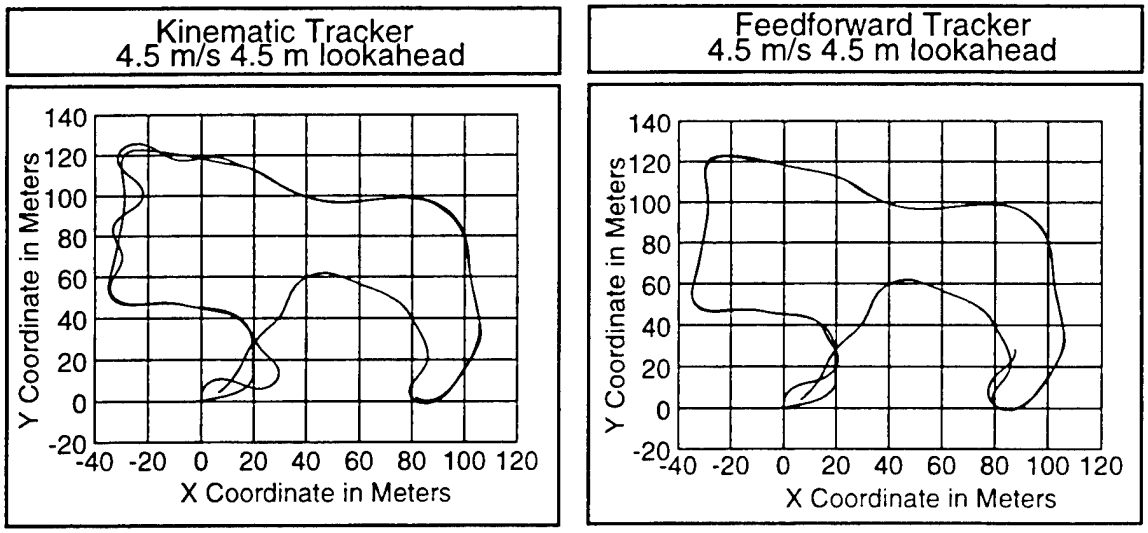


Figure 4: Performance of feedforward pure pursuit.

Each graph shows the target commanded trajectory and the actual trajectory followed under the algorithm used. Both paths are indistinguishable most of the time. Note, however, that the kinematic tracker goes unstable in the top left figure, whereas the feedforward one does not.

5. Adaptive Regard

The adaptive regard principle can be expressed as follows: There is no need to process data in regions where:

- The vehicle is already committed to going (dead zone).
- The vehicle cannot go (free zone).
- Subsequent images will provide better measurements (beyond detection zone).

because, in all cases, *there is no useful decision that a planner can make*.

In the first case, if an obstacle enters the dead zone, system failure is assured, so obstacles must be detected and avoided before they get this close. In the second case, processing such data is a complete waste of resources. In the third case, processing can be postponed until a future cycle.

If the vehicle cannot avoid obstacles at close quarters, not only is there no need for processing image geometry there, there is also no need for a planner to evaluate safety in its immediate vicinity in the map. The system planner evaluates trajectory safety only within the *planning window*. The algorithm for computing this win-

dow, based on an impulse turn maneuver, is given below:

```
/*
** Plan Window
*/
Pmax = speed * treact + rhomin;/* adaptive lookahead */
Pmin = Pmax - imaging_density * speed * cycle_time /* adaptive sweep */
```

An impulse turn is a turn from zero curvature to the maximum allowed curvature. It is the maneuver necessary to avoid a large obstacle without slowing or stopping. The imaging density is the average requested density of images on the groundplane. It must be maintained above 1.0 at all times to ensure adequate coverage of the environment. Later, the perception module will map this planning window into image space to allow it to extract the requested data.

6. Real-Time Latency Modeling

As a real-time system, a reliable measurement of time is required for the system to function correctly. In particular, the system needs to match images with vehicle states that may arrive intermittently and it needs to remember the commands that were issued to all actuators in previous iterations of the main control loop because the system cycles faster than the output latencies. This is accomplished with the following mechanisms.

- All input and output is time stamped.
- All input and output is stored in internal FIFO queues.
- All sensor latencies are modeled.
- All actuator latencies are modeled.

It is important to recognize that these FIFO queues do not introduce artificial delay. Their function is to *model* delay, which already exists in the hardware. Conceptually, all i/o goes through these queues.

Localization of Image Pixels: The smearing of the environment model which arises from incorrect models of the motion of the sensor during the image digitization process is called the *motion distortion problem.* All incoming vehicle state information is stored in a FIFO queue until it is needed to process images which arrive later. The state queue and the time tags are used to effectively register the ray through every pixel with a unique vehicle pose before its value is converted to world coordinates.

Localization of the Vehicle: While temporal registration of images and vehicle positions is a resolution matter, the raw delay of the vehicle position estimate is a separate accuracy concern. When the system uses the latest vehicle state as its estimate of position, the error involved in doing so is the velocity times the latency. Clearly a delay of only one second causes 5 meters of localization error in the vehicle position at 5 m/s and almost guarantees collisions with obstacles solely because the system does not know how close it is to the obstacle[1]. For this reason, an estimate of position sensor latency is used and fed forward just like any other parameter[2].

---

1. This is an almost intolerable situation, particularly when steering response is added to the picture, and it argues persuasively for tight system coupling. To ignore these issues at 5 m/s it to overestimate obstacle clearance by 5 meters, underestimate the distance required to turn by as much or more, and drive straight into anything in the way. However, the best that can be done is to model the latency if it cannot be reduced.
Roughly speaking, a 2 second delay from image to steering actuator makes 10 m/s speed impossible.
2. This is a practical measure because extant vehicle positioning systems and computing hardware have no consistent time standard. Ideally, the latency could be simply measured from the current time and the time tag of the image.

Localization of the Feedforward Vehicle: All output commands are stored in a FIFO queue and are used for the next several iterations in implementing command feedforward. When the system is generating the current command, there may be quite a few previous commands *still en route to the hardware* so a proper model must feed these pending commands into the model before the current one is used and must make decisions based on the last command to go out, because the system still has control over only that current command.

7. Feedforward Simulation

In order to evaluate safety, the tactical control module computes the response to candidate commands simulated forward in time until the detection zone is reached. The basic justification for this approach is an attempt to meet guaranteed response and localization simultaneously. In order to make sound decisions, the system must know *now* where the vehicle will be *many seconds later* if a particular command is issued now. It cannot "wait until it gets there" to measure its response because then it will be too late to react.

The solution of this problem is mathematically involved because it involves the solution of a coupled set of eight nonlinear differential equations which form the vehicle state space model. The system mechanizes these equations in real time. The equations are coupled because:

- Position at step i depends on attitude and speed at step i-1.
- Attitude at step i depends on steering response, propulsion response, and attitude step i-1.
- Steering response at step i depends on steering response and steering command at step i-1.
- Propulsion response at step i depends on attitude, propulsion response and propulsion command at step i-1.
- Attitude at step i depends on suspension state at step i-1.

7.1. State Space Terrain Following Model

By solving the equations in their true coupled form, the system can correctly simulate a vehicle driving along the side of a hill, for example. It correctly simulates the dependence of actual trajectory on the terrain itself, the speed, the actuator response characteristics, and the initial conditions.

The basic simulation loop can be written as follows. At each time step:

- **Simulate suspension:** determine attitude from terrain geometry and position.
- **Simulate propulsion:** determine new speed from command, state, and attitude.
- **Simulate steering:** determine angular velocity from steering and speed.
- **Simulate position:** dead reckon from linear and angular velocity and time step.

A mechanization diagram is shown in Figure 5 for these equations. In the figure, (x, y, z) represents the position of the vehicle in the navigation frame, (θ, ϕ, ψ) represents its attitude in terms of pitch, roll, and yaw angles respectively, V is the vehicle speed along the body y axis, α is the steer angle of the front wheels, β is the angular velocity of the body projected onto the body z axis, and the subscripts c and a represent commanded and actual quantities, respectively.

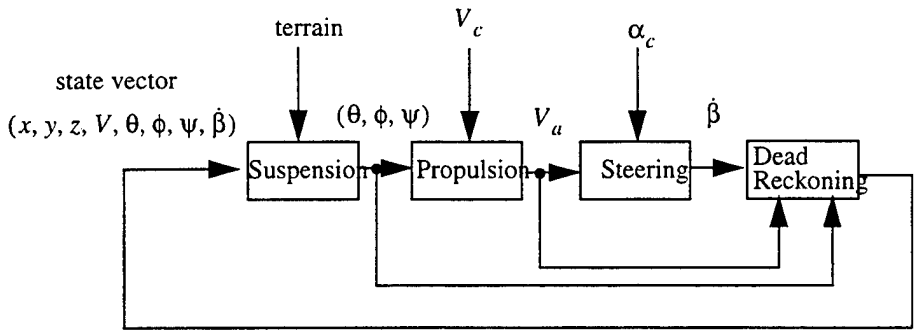


Figure 5: Feedforward simulator.

7.2. Reference Points

The positions of distinguished points on the body, called *reference points*, are maintained in navigation coordinates throughout the simulation. The kinematics transforms involved in doing this are documented elsewhere [26]. These points are at the same elevation as the bottom of the wheels and are positioned at the wheel contact points and the center of the axles. The processing of the reference points proceeds as follows for each cycle of the simulation:

- Convert coordinates from body to nav frame based on the vehicle position and attitude.
- Replace their z coordinates with those of the terrain at the same (x,y).

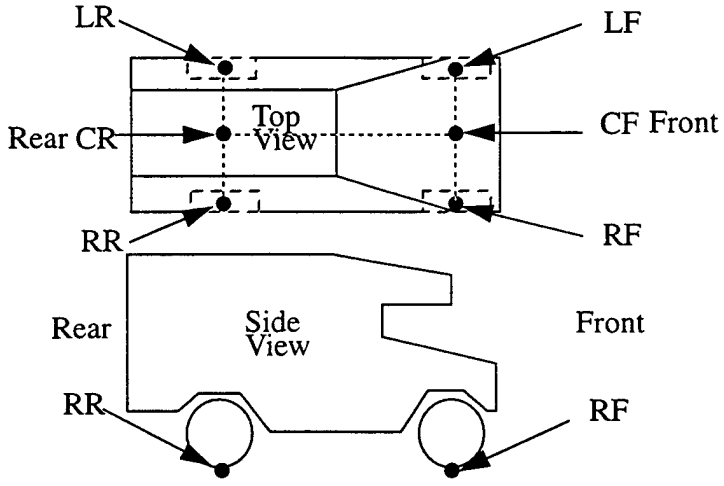


Figure 6: Reference points.

All aspects of both system state and hazardous conditions can be computed from the reference points and the terrain elevation under them. In this way, coordinate system transformation operations are reduced to an absolute minimum.

7.3. Suspension Model

Using the reference points only, a proper model would:

- compute the elevation "under" each wheel
- compute the deflections of each wheel from the minimum potential energy principle
- compute the positions of three points on the body
- compute the body attitude from these three points

which is a lot of somewhat costly work to do. Therefore, the suspension model is based on:

- a *rigid terrain assumption*
- a *rigid suspension assumption*
- a *locally planar terrain assumption*

The first assumption results from using the terrain map elevations measured by a sensor when the vehicle was not loading the terrain. The second occurs because the positions of the wheels relative to the body are taken as fixed. The third occurs because the elevations under the four wheels are used in forming two vectors even though they do not necessarily lie on the same plane.

The last problem in the process can be expressed as that of recovering an unknown rotation matrix from a few points which are transformed using it. The inverse RPY transform was computed in [26] for this purpose. The displacement transform from body coordinates to world coordinates for a z-x-y Euler angle sequence can be expressed as:

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \begin{bmatrix} (c\psi c\phi - s\psi s\theta s\phi) & -s\psi c\theta & (c\psi s\phi + s\psi s\theta c\phi) \\ (s\psi c\phi + c\psi s\theta s\phi) & c\psi c\theta & (s\psi s\phi - c\psi s\theta c\phi) \\ -c\theta s\phi & s\theta & c\theta c\phi \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}$$

Yaw and pitch can be determined from any vector known to be aligned with the body y axis:

$$\psi = \text{atan2}(r_{22}, -r_{12})$$

$$\theta = \text{atan2}(r_{32}, -r_{12}s\psi + r_{22}c\psi)$$

Roll can be derived from the world coordinates of a vector known to be aligned with the body x axis.

$$\phi = \text{atan2}(s\theta[-r_{11}s\psi + r_{21}c\psi] - r_{31}c\theta, (r_{11}c\psi + r_{21}s\psi))$$

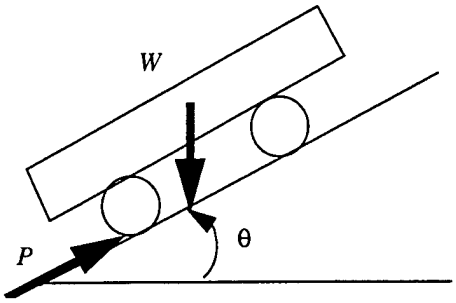
The above equations are an exact inverse kinematic solution. However, the system makes a valid *small pitch assumption* currently in order to reduce the trigonometric computations. The vectors are formed from the reference points. The algorithm is simply:

```
/*
** Simulate Suspension
*/
zleft = (lf[z] + lr[z]) / 2.0;
zrght = (rf[z] + rr[z]) / 2.0;
zfrnt = (rf[z] + lf[z]) / 2.0;
zrear = (rr[z] + lr[z]) / 2.0;
z = zrear + tire_radius;
pitch = atan2(zfrnt - zrear)/wheelbase;
roll = atan2(zleft - zrght)/width;
```

Using a linear deflection model, it can be shown that, since strain energy is squared in strain, the minimum potential energy principle of solid mechanics implies that a least squares solution to the deflections of the four wheels of an independent suspension is also the solid mechanics solution. The simple suspension model seems to work well enough as it is so that a more refined model was not implemented. Such a model however, would permit the representation of suspension overextension as a hazard. Also, the deviation of the four-wheel contact points from a plane is an equivalent model under a rigid suspension assumption and this would be simpler to implement.

7.4. Propulsion Model

Propulsion is currently modeled as a proportional controller with gravity compensation[1]. Under an assumption that the torque at the wheels can be directly actuated and that the terrain can generate any necessary shear reaction (no slip), Newton's law can be written along the body y axis as shown in Figure 7, which gives the plant model. A proportional controller using velocity feedback is configured as in the block diagram[2] shown in Figure 8.



$$\sum \vec{F} = M\vec{a}$$

$$P - W\sin\theta = Ma$$

$$\therefore a_{res} = \frac{P}{M} - g\sin\theta = a_{cmd} - g\sin\theta$$

Figure 7: Propulsion plant model.

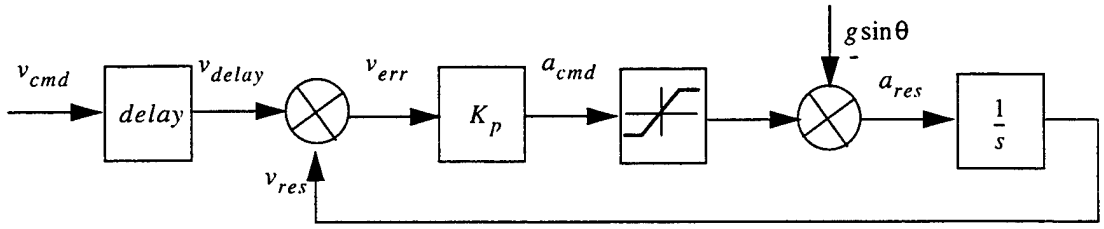


Figure 8: Propulsion servo.

The inverse of the proportional gain K_p is the time constant. A time constant of 5 seconds corresponds to the generation of 0.1 g command for a 5 m/s speed error. A limiter is added to reflect the fact that the output torque has some limit. One way to compute this limit is to choose a pitch angle corresponding to some grade that is the highest grade that the vehicle can climb. Under this model, the computer algorithm is the finite difference

1. This model is similar to the JPL speed controller except that throttle is actuated and not torque.
2. If, like most of us, the reader has forgotten Laplace Transforms, 1/s is an integral and s is a derivative.

version of the system differential equation, which is simply[1]:

```
/*
** Simulate Propulsion
*/
now = read_clock();
vdelay = queue_lookup(prop_queue,now - prop_delay);
verr = vdelay - vres;
acmd = verr / prop_time_constant;
if (fabs(acmd) > amax) acmd = amax * acmd / fabs(acmd);
ares = acmd - g * sin(pitch);
vres += ares * dt;
```

This model has several advantages over an ideal one. It will correctly cause the simulated vehicle to take several seconds to respond to a step input command, and it will even cause the simulated vehicle to back up on a steep grade and generally slow down or speed up, depending on the grade.

7.5. Steering Model

The steering model is involved because it must account for the nonlinear relationship between curvature and steer angle. This model is also a primary area of coupling between speed and attitude rate, so it is one of the most important elements of the model. The bicycle model of the steering kinematics is given in [26]. Typically, the steering wheel itself is actuated, so the loop must be closed back at the steer angle. This assumes only that the steering wheel to steer angle transform is linear. Either position or speed could be the controlled variable, but a proportional position loop is assumed[2]. Both the position and the speed are limited[3]. The complete dynamics can be expressed with the following block diagram:

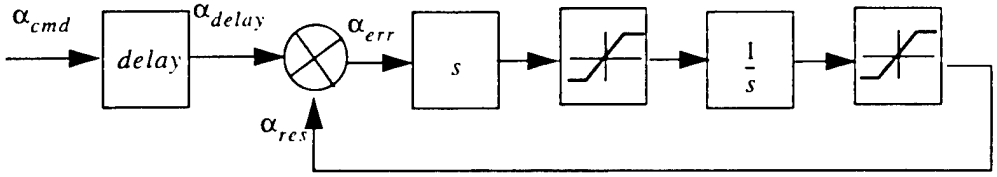


Figure 9: Steering servo loop.

Again the coding is straightforward:

```
/*
** Simulate Steering Dynamics
*/
now = read_clock();
alpdelay = queue_lookup(steer_queue,now - steer_delay);
alperr = alpdelay - alpres;
if (fabs(alperr/dt) > alpdotmax)
  alperr = alpdotmax * dt * alperr / fabs(alperr);
ares += alperr;
```

1. If we distinguish control algorithms from complete controllers, many actuator control algorithms can be implemented in just a few lines of code. The above situation is the rule, not the exception. Code is provided in order to illustrate that however sophisticated dynamic models may sound, implementing them is easy.
2. At this moment, a position loop is used because a speed loop requires speed feedback which is not available on some vehicles. It could be generated internally by differentiating the position feedback, but the speed limit is considered to be the overriding element of the delay anyway.
3. There may also be a trapezoidal command velocity profile. This amounts to an acceleration limit on the command.

These few lines of code make an enormous difference in the high-speed stability of the system.

7.6. Dead Reckoning

The 3D dead reckoning equations also form the system model in the Kalman filter, which is documented elsewhere [30]. The attitude rate about the body y axis is available from the instantaneous velocity and the instantaneous steer angle (or curvature) as follows:

$$\frac{d\beta(t)}{dt} = \frac{1}{L}\tan[\alpha(t)]\frac{ds}{dt} = \kappa(t)\frac{ds}{dt}$$

8. Tactical Planning

The system adopts the basic assumption that all hazardous conditions can be computed from the reference points. Hazards are represented as sampled time signals, which are parameterized by the commands to which they correspond. At any point in forward projected time, the system is designed to be robust to the absence of information because the *sampling* and the *occlusion problems* are intrinsic and inevitable.

8.1. Temporal State Interpolation

Field testing the system on typical rough terrain indicates that, *if occlusions are distinguished from poorly sampled regions* of the environment, then the need for sophisticated terrain map interpolation is questionable. Specifically, the degree of undersampling depends on the local incidence angle of pixels to the terrain and the need for interpolation directly implies that the terrain is mostly flat anyway. The reverse argument is more compelling. If a small vertical step in the terrain exists, then it must span several pixels due to its near normal incidence. Therefore, it cannot be undersampled.

These observations imply that it is unnecessary to interpolate an entire terrain map before it is used for planning computations. It can be done *on the fly*, and more importantly, it can be done *on the vehicle state vector.* The state vector alone supplies the information necessary to propagate the system state forward in time, and the terrain contact constraint can be modeled as a "disturbance" which modifies the vehicle attitude whenever the terrain under the wheels is known (which, in practice, is almost always).

The interpolation algorithm used in the system is an interpolation of both the state vector and the hazard signals in time, rather than an interpolation of the terrain map in space.

8.2. Hazard Identification

Three principal forms of hazards are incorporated:

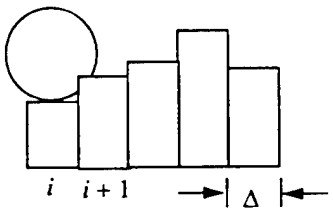
- **Unknown terrain:** Unknown map cells may arise from undersampling or terrain self occlusion. An overall information density merit is computed for each candidate command which is based on the number of wheel elevations that are known at each time step.

- **Collision of wheels with the body:** Wheel collision is computed as the slope of the elevations under each wheel as the body is propagated forward. In practice, this signal can be noisy if the sensor cannot supply the angular resolution and relative range accuracy necessary. At times, it is not used at all.

- **Collision of the terrain with the body:** Body collision is computed as the lost volume under the

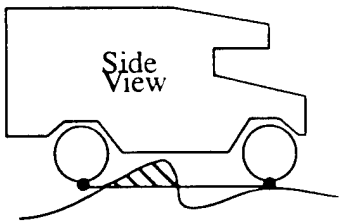
body normalized by the area under the body, so it is an expression of the average height under the body. The result is normalized by the number of map cells which were actually known.

- **Static instability:** Static stability measures the proximity of the vehicle to tipover. The *static stability margin* can be expressed as the remaining angle between the gravity vector and the support polygon formed by the wheel contact points (which is a rectangle in this case). Graphically, the margin is the minimum distance to any side of the support rectangle normalized by the height of the center of gravity above the terrain. The margin can be expressed as the maximum of the tangent of simple approximations to roll and pitch normalized by some maximum. The distinction between measurement as an angle or its tangent is irrelevant for small angles. One of the implications of this model is that the vehicle will turn into a steep grade so as to increase the stability margin. The system actually computes pitch and roll margins individually, and performs the maximum outside the hazard evaluator. This is a measure used to permit easy debugging. The system can navigate successfully over smooth terrain based on this hazard alone.



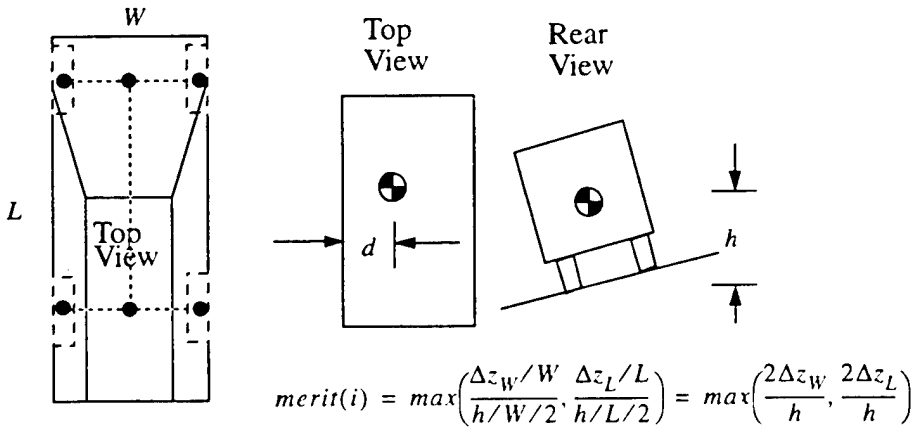
$$merit(i) = \frac{\frac{z_i - z_{i-1}}{\Delta}}{\left(\frac{\partial z}{\partial s}\right)_{max}}$$

Figure 10: Wheel collision.



$$merit(i) = \frac{\frac{1}{n}\sum_i z_i}{h_{max}}$$

Figure 11: Body collision.



$$merit(i) = max\left(\frac{\Delta z_W / W}{h/W/2}, \frac{\Delta z_L / L}{h/L/2}\right) = max\left(\frac{2\Delta z_W}{h}, \frac{2\Delta z_L}{h}\right)$$

Figure 13: Static stability.

8.3. Hazard Representation

In general, each hazard is represented on some normalized scale where the maximum value indicates certainty of danger and the minimum value indicates certainty of safety. A typical hazard signal might look like the following:

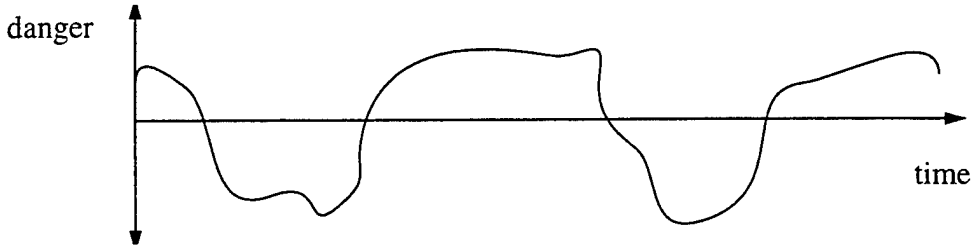


Figure 14: Hazard signals.

8.4. Hazard Arbitration

Conceptually, the first step to evaluate hazards is to generate a 3-D field of the form *merit(hazard,command,time)* because safety is different for each command, different for each hazard, and for a time signal. The obstacle avoidance hazard arbiter conceptually integrates this 3D field into a single decision. The first step of the process is to collapse the time dimension. This is done by computing the signal norm as follows:

$$merit = \left(\sum_i (merit(i))^{\alpha}\right)^{1/\alpha}$$

for some power α. In practice, however, a straightforward maximum was found to produce acceptable results. This step reduces the decision field to merit(hazard,command).

The next step is to collapse the hazard dimension. Hazards are considered independent because, for example, body collision is completely irrelevant to static stability. Therefore, a straightforward maximum applied across this dimension gives the worst unsafe condition for all dimensions of safety considered.

The final step is to reduce the command dimension, and this has been discussed in the context of joint arbitration of tactical and strategic votes. However, before the tactical vote vector is passed to the arbiter concerned, it is sometimes smoothed by *Gaussian filtering* in order to increase reliability. The idea is that, because the environment exhibits a degree of smoothness, it is appropriate at times for poorly rated trajectories to degrade the ratings of their spatial neighbors. In this manner, the impact of errors in both feedback and actuation will be less relevant because this mechanism will cause the vehicle to give obstacles a wide berth. At times, this mechanism is disabled depending on the spacing of the candidate trajectories investigated.

9. Speed Planning

Just like everything else, the speed which should be commanded of the vehicle depends on the local environment. It makes sense, for instance, for the vehicle to slow down when it encounters challenging terrain and likewise to increase speed when benign terrain is encountered.

The motivation for speed planning is that there is reason to expect that

- There is a way to measure system robustness.

- There is a way to relate robustness to speed in rough terms.

The principle used for a speed planner is to close a proportional control loop around the average predicted merit of all candidate commands because this average merit is low in dense hazard environments and high in regions free of hazards. This principle is complemented by the fact that higher speeds tend to lead to reduced maneuverability and hence lower reliability *for the same local terrain*. That is, robustness can be altered in a small region by adjusting speed alone.

Many aspects of performance are implicitly figured in the algorithm. For example, occlusion, resolution, map quantization noise, etc. are all implicitly expressed in the hazard signals, so the system will eventually reach equilibrium with respect to the most constraining sensor characteristic and an appropriate speed will emerge naturally.

When a planner gets confused, it tends to stay confused for a time and when it is relatively stable, it also tends to stay stable. This implies that there is a low frequency component to system robustness, which is long enough in time constant to exceed the vehicle response limits. Hence, a vehicle may be able to respond fast enough to change things before its too late. It should also be noted that the tactical controller is designed so that a panic stop (slamming on the brakes) is always guaranteed to avoid vehicle damage. The speed planner uses more graceful measures to improve performance.

A simple loop is configured as follows:

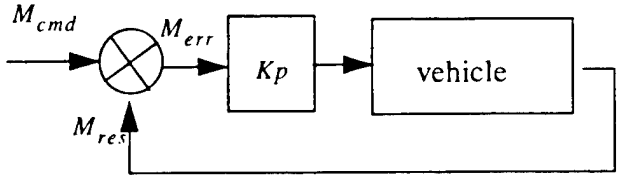


Figure 15: Merit servo loop.

This loop is implemented with a time constant that is relatively large so that the speed loop will respond only to the low frequencies in the overall merit. The results for a run of the system in a dense obstacle field at high-speed are indicated in Figure 16.

The vehicle was driven in simulation for 500 cycles at a nominal speed of 4.5 m/s at 3 Hz cycle time. When speed planning was on, the excursion was 703 meters versus 687 meters when it was off. The planner was considered to have failed when the merit fell below a threshold of 0.4 With speed planning on, "failure" occurred 4% of the time, whereas it occurred 6% of the time with speed planning off. More importantly, the seriousness of the failures with planning off were much worse than those with planning on. The system attempts to maintain a constant merit of 0.4 when the loop is closed. The figure shows that the deviation from 0.4 is much higher in the left figure. The peaks and valleys are wider in the left figure indicating sustained periods of confusion. In plain terms the algorithm makes it possible to drive farther more robustly.

An extra benefit of the speed planning loop is that it will automatically back the vehicle up and drive off in another direction when a panic stop is issued. The path planner supports this as well because it plans correctly for negative velocities. Therefore, it backs up "smartly" avoiding obstacles until a new alternative presents itself.

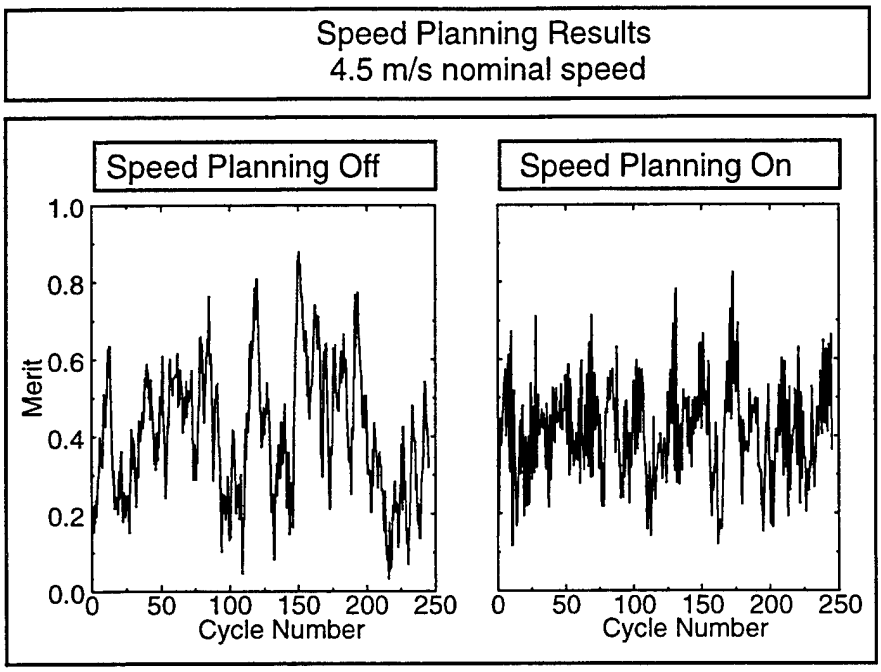


Figure 16: Performance of speed planning.

10. High-Speed Autonomy as an Optimal Control Problem

This section casts the problem of high-speed autonomy as an optimal control problem. The lowest level element of the system is the *control laws*. The control laws are responsible for coordinated control of all actuators. The control laws are implemented in a tightly coupled hierarchy. At this level of conceptualization, the system can be viewed as a multi-input, multi-output, feedforward state space controller.

10.1. State Space Controller

For *a linear system*[1], the conventional state space model of a system is the venerated two matrix equations:

$$\frac{d\underline{\mathbf{x}}}{dt} = A\,\underline{\mathbf{x}} + B\,\underline{\mathbf{u}}$$

$$\underline{\mathbf{y}} = C\,\underline{\mathbf{x}} + D\,\underline{\mathbf{u}}$$

Note in particular that the first equation is a differential one. The *command vector* $\underline{\mathbf{u}}$ includes vehicle steering and speed as well as demand signals to any sensor heads. The *state vector* $\underline{\mathbf{x}}$ includes the vehicle steering, speed, and the state of motion of the vehicle body and the sensor heads[2]. The *output vector* $\underline{\mathbf{y}}$ can be any func-

1. Our system is not linear. No real system is perfectly linear. Consider this a model for now, and the validity of the model to be an open question.

2. The state vector includes more information than the commands because the number of state variables is dictated by the order of the differential equation being modeled. The system equation is a matrix equation, so its order is the dimension of the A matrix, and this need have no relationship to the length of u. This corresponds to the fact that every integral requires an initial condition. On a conventional automobile the command vector u is of order 2, whereas the state vector may have 10 or more elements.

tion of both state and inputs, but in our case, it can be considered to be the state itself (so C is the identity and D is zero). These equations are often indicated in a block diagram as shown in Figure 17.

In this model so far, *the system transition matrix of the last section can be identified directly with the A matrix* and the equation that employs it to propagate the system state forward in time is the integrator. The transition matrix includes the steering model and the propulsion model. The B matrix can be identified with anything that modifies commands generated by the system before they get to the physical hardware. That is, the *B matrix can be identified directly with the delay queues.*

10.2. Hazard Model

Sometimes, the signals going to the plant can be further distinguished from the information which is ultimately of interest. In our case, the primary quantity of interest is vehicle safety. This can be indicated with a *reference input* or demand vector $\underline{\mathbf{d}}$. Also, there may be quantities which influence the state of the system over which there is no control at all. Let these *disturbances* be called $\underline{\mathbf{u}}_d$. Disturbances can be placed anywhere that makes sense in the diagram and they may or may not be modeled. Further, the C matrix can be reintroduced and the output considered to be the safety of the vehicle expressed in terms of the hazard signal vector. The new system model is shown in Figure 18, where a single box for the *state space model* S has been generated by grouping certain elements together. Such a system implemented without feedforward would be a regulator that attempts to keep the hazard vector at 0.

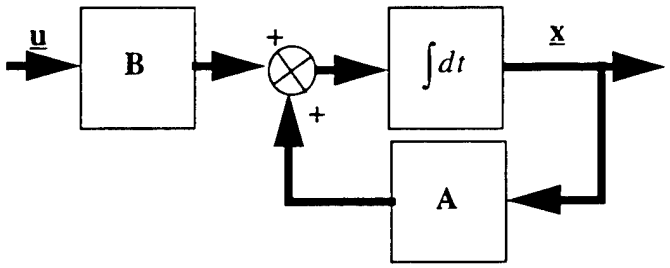


Figure 17: Linear system block diagram.

The disturbances can be considered to be anything that affects the system but that is generated outside the system. In our case, the requirement that the vehicle remain on the terrain surface arises from the fact that the terrain is incompressible (by assumption) and that it generates forces that prevent the vehicle from sinking. Thus, *the disturbances can be identified directly with the suspension model.* The *output vector is the hazard signal vector* and the *C matrix is the hazard evaluation functions*, which map vehicle states and the terrain geometry onto estimate of hazards. Notice that the hazards are time signals and that there are many kinds. Each kind of hazard is one element of the output vector.

This model maps very directly onto the tactical controller. Other control loops can be considered to be subsumed in the diagram as well. Sensor head control is a regulator where the reference input is the deviation of the range window from image dead center. The strategic controller is a regulator that attempts to maintain the heading error against the global path at zero. A complete diagram is far too detailed to draw because:

- The state vector is of order close to 10.
- Many of the matrices are divisible into blocks.
- Coordinate system transforms are complicated and are a major source of coupling and nonlinearity.

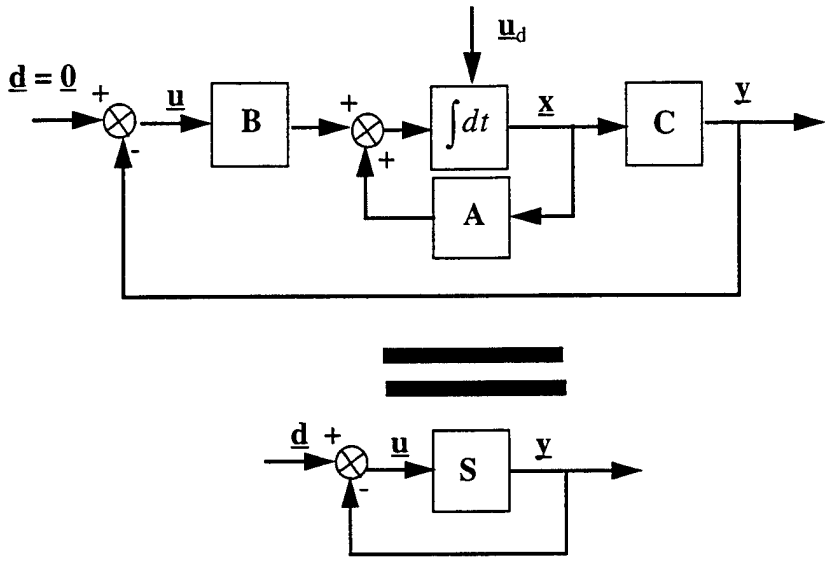


Figure 18: State space model.

10.3. Feedforward Optimal Controller

Feedforward is employed for two distinct reasons. First, it imparts stability to many of the algorithms - particularly the strategic controller and the tactical controller[1]. Second, the process by which safety is predicted is also a feedforward process.

The system attempts to guarantee safety by *predicting* into the future the relationship between future safety and current command options. Then it acts now based on future consequences. It does this by using a *state space simulator.* This feedforward element can be represented by duplicating the state space model and indicating that it is an estimate with a super hat diacritical mark. The fact that an optimal trajectory is chosen can be represented by replacing the summing point with a more complex minimum transformation, as shown in Figure 19.

The state space model maps easily onto the tactical controller. The tactical controller is a kind of suboptimal controller that *attempts to minimize the norm*[2] *of the hazard vector.* This is *guaranteed safety* again in controls language.

Formally, the system is a suboptimal solution to the following optimal control problem:

$$min[L(\bar{y}(t))] = \sqrt{\left[\sqrt{\int_0^t y_i(t)^2}\right]^T \left[\sqrt{\int_0^t y_i(t)^2}\right]}$$

$$\frac{d\underline{x}}{dt} = A\,\underline{x} + B\,\underline{u}$$

$$\underline{y} = C\,\underline{x} + D\,\underline{u}$$

1. It has not been mentioned yet, but feedforward prevents "waffling" in the steering arbitrator because once a slight turn is commanded once, it affects all downstream computations by biasing all subsequent decisions to reflect the distance the steering wheel turns for that cycle. Each cycle's output is a partial commitment, so the tactical controller can only sit on the fence with respect to a particular obstacle for one cycle. Feedforward leads to a more confident controller because it recognizes when a partial commitment has been made to turn in a certain direction.
2. The norm is computed by integrating first over time and second over the dimensions of hazard space.

where A models the steering and propulsion dynamics, B models the communications delays, and C models the terrain contact constraint as well as the mapping from system state onto hazards. The rms time integral of the hazards is generalized to the signal α norm given by:

$$\sqrt{\int_0^t y_i(t)^2} \Rightarrow \left(\int_0^t y_i(t)^\alpha\right)^{1/\alpha}$$

and the vector length is generalized to an α norm over hazard space:

$$L_\kappa(y) \Rightarrow \left[\sum_i (y_i)^\alpha\right]^{1/\alpha}$$

The formal solution to an optimal control problem is obtained from the *Pontriagin Maximum Principle* (of which the more familiar *calculus of variations* is a special case). In general, the optimal control problem generates a set of simultaneous partial differential equations with boundary conditions which are very difficult to solve in real time.

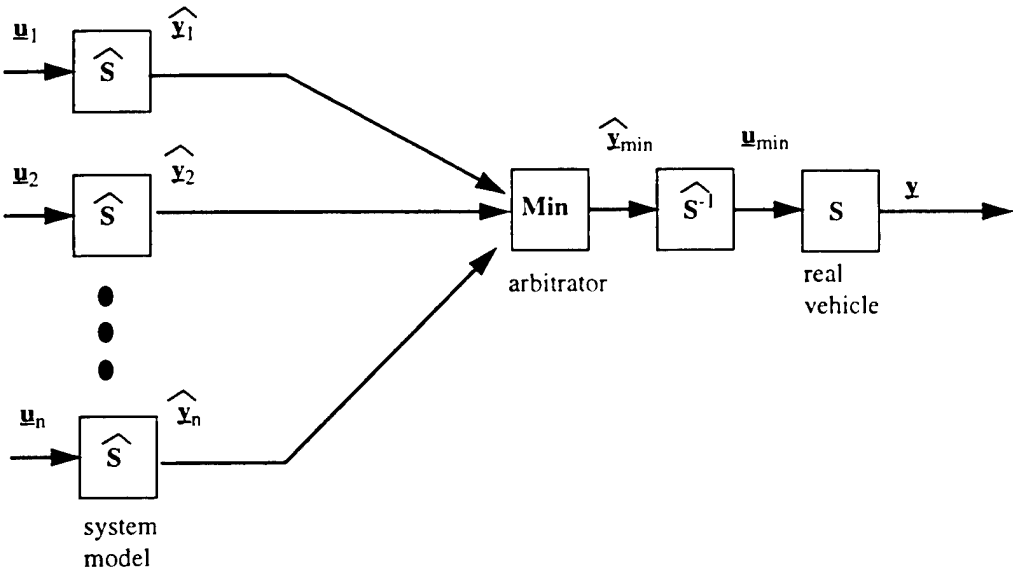


Figure 19: Control laws.

The system model amounts to a complicated differential equation constraint. The satisfaction of this constraint is generally very difficult to achieve. The set of trajectories, which satisfies the system model equations and which maintain the vehicle in contact with rigid terrain, is called, in optimization lingo, the *feasible set.*

The system satisfies this constraint *by construction* through feedforward. The inverse system model in the previous diagram is never evaluated explicitly - the system simply remembers the correspondence of commands to response trajectories and inverts this list of ordered pairs. In this manner, the system *plans in actuation space.*

The system solves the optimization problem, effectively, by sampling the feasible set of trajectories at some practical resolution that ensures adequate coverage of the set, and then by choosing the trajectory with the best value of the functional $[L(\bar{y}(t))]$.

Bibliography

[1] O. Amidi. *Integrated Mobile Robot Control.* Robotics Institute Technical Report CMU-RI-TR-90-17, Carnegie Mellon University. 1990.

[2] M. G. Bekker. *The Theory of Land Locomotion.* The University of Michigan Press. 1956.

[3] M. G. Bekker. *Off-the-Road Locomotion.* The University of Michigan Press. 1960.

[4] R. Bhatt, L. Venetsky, D. Gaw, D. Lowing, A. Meystel. A Real-Time Pilot for an Autonomous Robot. *Proceedings of IEEE Conference on Intelligent Control.* 1987.

[5] W. L. Brogan. *Modern Control Theory. Quantum Publishers.* 1974

[6] R. A. Brooks. A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control", *Proceedings of IEEE International Conference on Robotics and Automation.* 1987.

[7] B. Brumitt, R. C. Coulter, A. Stent. Dynamic Trajectory Planning for a Cross-Country Navigator. *Proceedings of the SPIE Conference on Mobile Robots.* 1992.

[8] T. S. Chang, K. Qui, and J. J. Nitao. An Obstacle Avoidance Algorithm for an Autonomous Land Vehicle. *Proceedings of the 1986 SPIE Conference on Mobile Robots*, pp. 117-123.

[9] M. Daily. Autonomous Cross Country Navigation with the ALV. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 718-726. 1988.

[10] E. D. Dickmanns. Dynamic Computer Vision for Mobile Robot Control. *Proceedings of the 19th International Symposium and Exposition on Robots*, pp. 314-27. 1990.

[11] E. D. Dickmanns, A. Zapp. A Curvature-Based Scheme for Improving Road Vehicle Guidance by Computer Vision. *Proceedings of the SPIE Conference on Mobile Robots.* 1986.

[12] J. C. Dixon. Linear and Non-Linear Steady State Vehicle Handling. *Proceedings of Instn Mechanical Engineers.* Vol. 202, No. D3. 1988.

[13] R. T. Dunlay, D. G. Morgenthaler.. Obstacle Avoidance on Roadways Using Range Data. *Proceedings of SPIE Conference on Mobile Robots.* 1986.

[14] D. Feng. *Satisficing Feedback Strategies for Local Navigation of Autonomous Mobile Robots.* CMU Ph.D. Dissertation at CMU. May. 1989.

[15] D. Feng, S. Singh, B. Krogh. Implementation of Dynamic Obstacle Avoidance on the CMU Navlab. *Proceedings of IEEE Conference on Systems Engineering*, August. 1990.

[16] U. Franke, H. Fritz, S. Mehring. Long Distance Driving with the Daimler-Benz Autonomous Vehicle VITA. *Proceedings PROMETHEUS Workshop.* 1991.

[17] J. Gowdy, A . Stentz, and M. Hebert. Hierarchical Terrain Representation for Off-Road Navigation. *In Proc SPIE Mobile Robots.* 1990.

[18] D. Langer, J. K. Rosenblatt, M. Hebert. A Reactive System For Off-Road Navigation. CMU Tech Report

[19] M. Hebert and E. Krotkov. Imaging Laser Radars: How Good Are They. *Proc. IROS 91.* 1991.

[20] M. Hebert. "Building and Navigating Maps of Road Scenes Using an Active Sensor. *In Proceedings IEEE conference on Robotics & Automation.* 1989.

[21] M. Hebert, T. Kanade, and I. Kweon. "3-D Vision Techniques for Autonomous Vehicles. Technical Report CMU-RI-TR-88-12, The Robotics Institute, Carnegie Mellon University. 1988.

[22] B.K Horn and J. G. Harris. Rigid Body Motion from Range Image Sequences. *Image Understanding*, Vol 53, No 1. 1991

[23] R. Hoffman, E. Krotkov. Terrain Mapping for Outdoor Robots: Robust Perception for Walking in the Grass. *Proc. IEEE International Conference on Robotics and Automation.* 1993.

[24] D. Keirsey,D. Payton, J. Rosenblatt. Autonomous Navigation in Cross-Country Terrain. *Proceedings of Image Understanding Workshop*. 1988.

[25] A. Kelly, A. Stentz, M. Hebert. Terrain Map Building for Fast Navigation on Rough Terrain. *Proceedings of the SPIE Conference on Mobile Robots*. 1992.

[26] A. J. Kelly. *Essential Kinematics for Autonomous Vehicles*. CMU Robotics Institute Technical Report CMU-RI-TR-94-14.

[27] A. J. Kelly. *Modern Inertial and Satellite Navigation Systems*. CMU Robotics Institute Technical Report CMU-RI-TR-94-15.

[28] A. J. Kelly. *A Partial Analysis of the High-Speed Autonomous Navigation Problem*. CMU Robotics Institute Technical Report CMU-RI-TR-94-16.

[29] A. J. Kelly. *Adaptive Perception for Autonomous Vehicles*. CMU Robotics Institute Technical Report CMU-RI-TR-94-18.

[30] A. J. Kelly. *A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles*. CMU Robotics Institute Technical Report CMU-RI-TR-94-19.

[31] A. J. Kelly. *An Intelligent Predictive Controller for Autonomous Vehicles*. CMU Robotics Institute Technical Report CMU-RI-TR-94-20.

[32] A. J. Kelly. *Concept Design of A Scanning Laser Rangefinder for Autonomous Vehicles*. CMU Robotics Institute Technical Report CMU-RI-TR-94-21.

[33] In So Kweon. *Modelling Rugged Terrain by Mobile Robots with Multiple Sensors*. CMU PhD Thesis. 1990.

[34] T. Lozano-Perez, and M. A. Wesley. An Algorithm for Planning Collision Free Paths Among Polyhedral Obstacles". *Communications of the ACM*, Vol. 22, Num. 10, October 1979, pp. 560-570.

[35] M. Marra, R. T. Dunlay, D. Mathis. Terrain Classification Using Texture for the ALV. *Proceedings of SPIE Conference on Mobile Robots*. 1988.

[36] L. S. McTamaney. Mobile Robots: Real-Time Intelligent Control. *IEEE Expert*, Vol. 2, No. 4. 1987.

[37] H. P. Moravec. The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, Vol. 71, Num 7, July 1983.

[38] K. Olin, and D. Tseng. Autonomous Cross Country Navigation. *IEEE Expert*. pp. 16-30. 1991.

[39] D. A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*. Vol. 3, No. 1. 1991.

[40] J. Rosenblatt,D. Payton. A Fine-Grained Alternative to the Subsumption Architecture. *Proceedings of the AAAI Symposium on Robot Navigation*. 1989.

[41] S. Singh et. al. *FastNav: A System for Fast Navigation*. Robotics Institute Technical Report CMU-RI-TR-91-20, Carnegie Mellon University. 1991.

[42] S. Shafer, A. Stentz, C. Thorpe. An Architecture for Sensor Fusion in a Mobile Robot. *Proceedings of IEEE International Conference on Robotics and Automation*. 1986.

[43] H. C. Stone. *Design and Control of the Mesur/Pathfinder Mocrorover*. JPL Technical Report. 1993.

[44] D. H. Shin, S. Singh and Wenfan Shi. A partitioned Control Scheme for Mobile Robot Path Planning. *Proceedings IEEE Conference on Systems Engineering*. Dayton, Ohio. 1991.

[45] A. Thompson. The Navigation System of the JPL Robot. *Proceedings of the International Joint Conference for Artificial Intelligence*. 1977.

[46] B. Wilcox et al. A Vision System for a Mars Rover. *Proceedings SPIE Mobile Robots II*. Cambridge Mass., pp. 172-179. 1987.

Sensor Fusion For Autonomous Navigation Using Neural Networks

1. Introduction

Some of the most successful work to come out of the UGV group has included road following using CCD cameras [10] and cross country (XC) navigation using a scanning laser range finder [7]. The long-term goal of each of these projects has been to engineer a vehicle capable of performing fully autonomous tasks in a given environment.

But current systems have significant limitations. A single CCD camera is sufficient for road following, but not for extended road-driving tasks; a laser range finder is sufficient for obstacle avoidance, but not for cross-country navigation in "interesting" terrains.

1.1. Fully Self-Contained Autonomous Navigation

In both the road following and the cross country navigation tasks, as well as in many other tasks (such as space exploration), *fully self-contained* navigation is either very desirable or entirely necessary. Although we can enhance capabilities with "top-down" data such as maps, the techniques that have worked so well to date for road-following and obstacle avoidance have been "bottom-up" techniques, which take the rawest of locally available data and turn this into the necessary motion control commands. Inherent in this concept is rich sensing. Thus, our goal is to integrate multiple sensors in order to achieve capabilities not previously attainable in autonomous navigation systems.

1.2. Prior Work and Our Approach

The approach taken in our research is informed by much of the work in cross-country navigation [1][7], work in neural networks for navigation [2][8][9][10][13], and work in modular neural networks [3][4][5][6][12].

Current experiments with sensor fusion use backpropagation neural networks [11]. The basic elements of a neural network are nodes and connections. Nodes have activation levels, and connections have weights. The nodes are frequently organized into three groups: inputs, outputs, and hidden units. The network learns by having a pattern put into the input nodes and adjusting the weights of all of the connections until the output nodes show the desired output pattern.

Why are neural networks appropriate for this task? For an excellent discussion of the suitability of neural networks for the general navigation task see [10]. The primary issue is that we do not necessarily know all of the features in sensor space that affect our navigation decisions. Neural networks, when properly trained, can automatically select and weight the most important features of an environment, with the added bonus of being able to tackle a radically different environment with just a change of training data.

The architectures of the networks we examine in this work are of two flavors: monolithic networks and modular networks. In the typical monolithic neural network approach, we give the network all of the sensor data as inputs and allow the network to develop internal representations (by adjusting weights between nodes), which allow it to perform adequately within the context of the experiences we expose it to.

However,the internal representations that the neural network generates may be difficult to interpret. The network may even cue on features that are artifacts of our training data and not central to the task. What if it is

know ahead of time that some feature in the sensor space is useful for navigation? For example, it is known that the edges of the road are good clues to use in on-road navigation. A typical neural paradigm might result in a network learning to find the edges of the road. Or it might not. Exactly how much of our resources (time, memory, computing power, patience) are we devoting to learning what we already know?

These issues have led to examine modular neural networks as an alternative. The proposed modular architecture addresses the integration of a priori knowledge and may prove to be more flexible than a monolithic approach as the complexity of the tasks increase. The modular system is the Modular Architecture Multi-Modal Theory network (MAMMOTH). MAMMOTH is both a network architecture and a training paradigm.

A MAMMOTH system consists of two segments, the feature level and the task level. The feature level is a set of Feature Neural Networks (FeNNs) trained to recognize specific features in whatever sensor modality serves as their input source. Thus, there might be one or more networks in the feature level devoted to known-to-be-useful features in the color video modality. Other FeNNs might recognize features in laser range finder sensor space or in the input space of any modality.

On top of this feature level is the task network which unites the feature networks, the Task Network (Task Net). The task level uses the information from the feature level networks' hidden layers as inputs to a network trained to perform the navigation task. For a sketch of a MAMMOTH network see Figure 2.

2. The Task at Hand

Our task is to achieve autonomous navigation with the HMMWV. The vehicle must wander safely around the terrain-avoiding obstacles (but not heading to particular goals). The system should choose an appropriate steering direction (throttle is not computer-actuated). The HMMWV has all of its computing, power, and sensing on-board. We use both primary sensors: a color video camera and a scanning laser range sensor.

Most of the work was tested in one of two environments: on a square kilometer of a "virtual" 2.5D world or at 2-kilometer by 2-kilometer outdoor testing site on top of an old slag heap. Alsim, our simulator [7], consists of a map with elevation and color (RGB) in each cell. The initial elevations are generated with a triangle based fractal, and coloration is based on a similar fractal. This creates a world with varying elevation and patches of "vegetation" (a.k.a. green stuff). We can add onto this world a variety of colored geometric shapes and dirt or paved roads.

The real world slag heap has sections that range from very sparsely vegetated to untraversible due to trees and thick bushes. There are natural and man-made ridges and mounds of earth, cliffs, large semi-permanent puddles, and remnants of the foundations of a few buildings. A dirt access road runs through a large section of the slag heap (looping around to almost 4k total length). A rough estimate would be that less than half of the total surface area of the slag heap is traversible with the HMMWV. Most current systems are even more limited in the sections they can reach because they do not use multiple sensors and correspondingly cannot deal well with vegetation.

In both Alsim and real world experimentation (and especially in the real world), speed of processing is important, and in order to keep the whole control loop going at slightly faster than a convenient (if slightly arbitrary) goal of 1 Hz, we reduced the images from their original high resolution to lower resolutions. This cut down on processing time for the neural network simulations. Fortunately, in order to achieve the goal of proof of concept for the sensor fusion mission, we did not need to detect *every* obstacle which could be harmful to the HMMWV. We counted on the human safety driver to keep the vehicle from harm due to obstacles below sensor resolution. We felt safe in using laser images that were 16 x 64 pixels, and camera images that were 15 x 20 in each of the red, green, and blue bands (reduced from 64 x 256 and 480 x 640 respectively). This meant that most of our runs were stopped at some point when the vehicle encountered an obstacle it physically could not see.

3. Network Architecture

The neural networks, also, had to be tailored for execution in real-time. Generally, this was not a problem because the biggest implication for the networks was that the number of hidden units (see [11] for a full explanation) had to be kept to as few as possible to accomplish the task. But neural networks are function approximators, creating a mapping from the input to the output. The number of hidden units roughly corresponds to the dimensionality of the functionthat is fit to the data. If a high-dimensional function is used for a data set which could be represented better by a low dimensional function, it may fail to extrapolate from or (worse) interpolate between the provided data points. Thus, you want to use as few hidden units as necessary.

3.1. Monolithic architectures

Two monolithic neural network architectures were developed which performed almost identically. Most of the time a simple thre-layer feed-forward network with 1860 inputs from the laser and the CCD camera was used, 5 hidden units, and 11 output units. The outputs and hidden units used were based on results from Pomerleau [10] and from experimentation. The output activations were done in a Gaussian form, such as Pomerleau used, so that the activation was highest on the node corresponding to the desired steering direction and sloped off to either side. An additional hidden unit of five nodes was sometimes added between the first hidden layer and the outputs in order to make the complexity of connections similar to those found in the MAMMOTH network used. See Figure 1 for a diagram of this network.

To train the monolithic network, we gathered a training set of 300 exemplars (an exemplar consists of an input image to output direction pair). Then the network was trained in 200 to 500 epochs (an epoch is one pass through the whole training set in batch training [11]) until it performed adequately for our tasks.

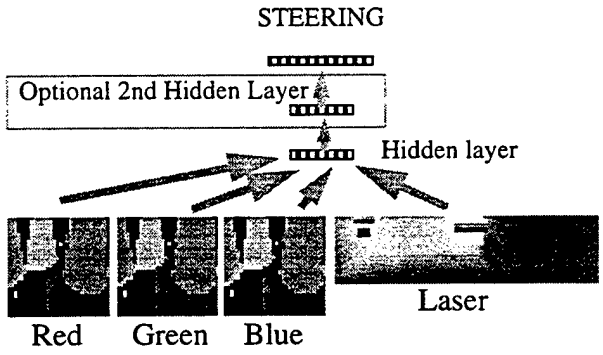


Figure 1: Monolithic navigation network.

3.2. The specific MAMMOTH architecture

The same MAMMOTH architecture was used for all of the modular network experiments, which had the same number of inputs and outputs as the monolithic network, and which had roughly the same number of connections. In this case, all of the CCD inputs (in red, green, and blue) are connected to one set of hidden units which are connected to one set of outputs. This feature network, or FeNN, was trained to find features relevant to driving in the color image space. Another FeNN connected the range image inputs to a second set of hidden units and to another output. The task network with a set of hidden units and its own output layer sat above this and was connected to the hidden units of the two FeNNs (Figure 2).

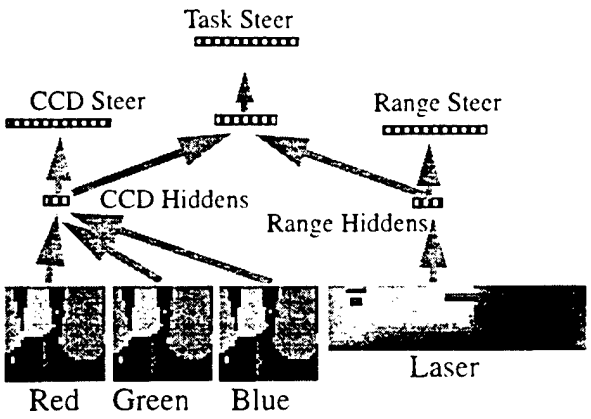


Figure 2: MAMMOTH network for navigation.

The interesting aspect of the FeNNs is that they are trained before the task network is trained, and then their weights are frozen. The task network is then trained with the hidden units of the FeNNs as its inputs. If the FeNNs are well designed and the training sets for the FeNNs are constructed carefully, you can force these hidden units to represent a narrow portion of the input space. For our experiments the task network was trained with the exact same training sets as the monolithic networks.

4. Experiments and Results

The two experiments in Alsim took the road following work of Pomerleau [10] a step further by adding obstacle avoidance with the laser range sensor. Likewise, our two real world tests augmented obstacle avoidance with CCD camera data.

4.1. Alsim: Stationary road-colored obstacles

The first experiment involved following a road while avoiding obstacles. The obstacles were stationary and black, which was the same color as the road. The vehicle had to use both sensors (color video and laser range) to accomplish this goal. Figure 3 shows a map of the simulated terrain. There are block-shaped obstacles at points A, B, & C. A black pit is at point D. The road we want the vehicle to follow is the paved road with the line down the middle. There is a dirt road (brown) running east-west, and various geometric obstacles off-road.

Both monolithic and MAMMOTH networks were used for this test. We collected training exemplars once per second. For the monolithic network's training set and the Task Net of the MAMMOTH network, the vehicle was driven (using the mouse to steer) over the road several times in each direction. Additional exemplars were gathered by driving off the road (with data collection off) and then driving back onto the road; these exemplars were necessary so that the simulated vehicle could recover from small navigational errors. Meanwhile, the vehicle was steered so as not to hit any of the three obstacles on the road and to drive on the road when possible.

For the monolithic network we were able to train just with this data. However, for the MAMMOTH network, we first trained the FeNNs. One FeNN was for road-navigational features using the simulated CCD camera and the other FeNN was for obstacle-avoidance-navigational features using the simulated laser range finder.

Alsim gave us the power to easily make training sets for FeNNs. The nature of an FeNN is that we want it to learn internal representations (weight patterns) that represent a narrow and known set of features. In Alsim we can force an FeNN to learn only what we want by creating worlds whose only features are those we wish to

learn. To train an FeNN to recognize low-level video features of roads appropriate for navigation and not to key off of extraneous image features, we make a world in which the only feature is a road. Likewise, to train an obstacle-avoidance FeNN, we make a world in which the only features are large obstacles to be avoided.

For the MAMMOTH network, 300 training exemplars were gathered in both the road-world and the obstacle-world. The respective FeNNs were trained with these and then the weights to the outputs of these FeNNs were frozen. Finally, the same training set used on the monolithic network was used to train the Task Net part of the MAMMOTH architecture.

Performance was excellent on this task for both networks. For both training and testing, the speed of the simulated HMMWV was held perfectly constant, and the vehicle was able to stay on the road and avoid obstacles for over an hour and a half, which corresponds to about an hour at a slow speed (3mph) in the real world. Another option of Alsim allowed us to "warp" the vehicle to a random location and orientation in the simulated world, and three times out of four the vehicle resumed traveling on the road when it intersected the road again. About every fourth time the vehicle would intersect the road at an almost right angle which was sufficiently different from what was in the training set , and it would respond improperly and drive straight across the road.

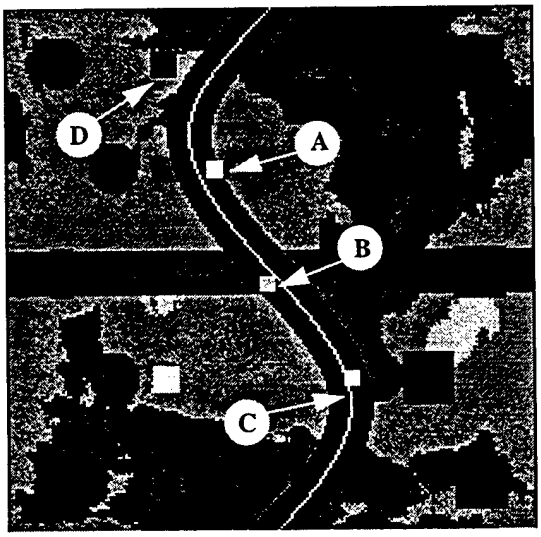


Figure 3: Map of the simulated world.

4.2. Alsim: brightly colored obstacles

Another simulator task highlights one of the pitfalls that hinders researchers using neural networks, and the results suggest two solutions. The goal was again to stay on the road and to avoid the static obstacles, but this time the three obstacles were colored blue, green, and yellow from north to south along the road. Training sets were gathered in the same manner as in the earlier task.

The results were similar to those of the previous section. The particular training set for this task produced slightly sloppier driving habits in the simulated HMMWV, and this was evidenced by the occasional (maybe 1 in 8 occurrence) of driving fairly widely around the blue box at point A and then heading straight into the pit at point D. This behavior was bad. However, the explanation and fix are simple. This happened because the vehicle had not driven so far to the left on the road near point A before and thus had not seen the pit very well before. The pit and the walls of the pit were black - the same color as the road. If the vehicle had just avoided the block at point A widely, it would see the black of the pit better than the black of the road and head that way. The fix is simply to add examples of avoiding pits to the training set.

A more interesting result occurred when the networks which had been trained with the blue, green, and yellow blocks, were tested on a simulated world which had three bright red blocks at the same locations. The vehicle went off the road and into the pit much more frequently. With the monolithic network there was at least a two in three chance of going off of the road, and with the modular network there was approximately a one in three chance. The implication is clear. The person training the network had correctly driven more closely around the blue box at point A than the yellow box at point C in the training. And both networks had learned to go left around blue boxes more tightly than around yellow boxes. With the red blocks, the network drove more loosely around the block at point A, and saw the pit that much more often. Another symptom was that the vehicle frequently went off of the road at point B. There was an almost 100% occurrence with the monolithic network and at least a 50% chance with the modular network.

Both of these problems have the same explanation: the yellow block in the first colored-block world was the only block with a red component in RGB. The network learned to go wide around the yellow block since it had to go around it on the left to stay on the road and there was also a left turn immediately after the block on the road. This made the network go widely to the left of the block at point A and often to the left of the block at point B.

A simple fix is training sets that have examples of blocks of every color on both sides of the road. But note that although the training set for the MAMMOTH task network was identical to that for the monolithic network, it helped that the lower level representations were strictly separated into obstacles in laser range space and representations for road following. By designing our FeNNs to represent narrowly what we want, we decreased the amount that we keyed on incidental features.

4.3. Real world: additive obstacle test

If everything in the world were rigid, we could avoid all the obstacles [7]. However, sometimes we must drive over the vegetation when we have no other decent route. The laser range finder alone cannot find vegetation, so we must use an additional sensor, the color video camera.

The first task has a simple goal: don't hit big solid obstacles and also don't hit vegetation. The first task is called the additive obstacle task because the additional sensor only adds new things to avoid.An area of the slag heap containing an access road that often had vegetation and/or earth mounds on one or both sides was used for training and testing.

We used the same network architectures as for the simulated tasks. It is clear that generating appropriate training sets is significantly more difficult. For the Task Net part of the MAMMOTH network and for the monolithic network, we drove the HMMWV around the terrain. This would have been simple, except we needed examples of recovering from mistakes, which meant we had to make mistakes (with data collection off), and mistakes are that much trickier in a 5-ton truck than in a simulated 5-ton truck. Consequently, speed varied, and the training set was less rich in recovery examples.

But training the FeNNs was the real challenge. In the real world we cannot simply train in a world that contains the narrow set of features we wish to detect. The laser range FeNN was trained to avoid obstacles, as it had been in Alsim. There were several large mounds of earth sufficiently far from vegetation that we used as our prototypical obstacles, and we trained by avoiding these mounds as we had done in Alsim.

Another FeNN was trained to drive around avoiding vegetation. In the real world, though, images containing vegetation often contain other things; additionally, vegetation in the real world is not nearly as homogeneous as any feature we might want to recognize in the simulated world. We had to develop a training set that contained a rich set of examples of the HMMWV steering away from all of the types of vegetation we might encounter.

Given these difficulties, both FeNNs we trained worked quite well, and the monolithic network and the Task Net performed even better. In fact, our two main problems were that there were obstacles that simply didn't

show up at the resolutions we used in sensor space (such as sharp steel bars and jagged cinder blocks) and that there were culs-de-sac at the slag heap.

Both FeNNs would allow the HMMWV to drive for runs of up to 0.6 miles before we ended up in a cul-de-sac or had something dangerous right in front of us that could not be seen at low resolution. Another failure mode was the result of uneven ground or territory not covered in the training sets for the FeNNs. The most frequent failure mode for the CCD FeNN was driving right up to a cliff that was the same color as the road. Likewise, a common failure mode for the laser range FeNN was driving into low vegetation that got progressively more dense until we gradually appeared in an area where the vegetation was dense enough that no correct steering direction could be determined from range images.

Combining the two sensor modalities had the desired results. Typical runs of the HMMWV were around 1 mile or more before we encountered a failure mode. Both the monolithic network and the MAMMOTH network performed very well, though there is not yet enough data for a significant quantitative comparison.

4.4. Real world: conflicting obstacle test

The final test was one that we had just begun to tackle on the real vehicle. The location of the test was a section of roughly 1600 square meters on the slag heap, which has numerous man-made mounds of earth spaced just a bit wider than is necessary for the HMMWV to drive through. The mounds of earth were approximately 3 feet high and 5 feet in diameter, and between them vegetation grows out of control. The task was to drive the HMMWV through this obstacle course. The laser range sensor generated images in which the vegetation looked like the mounds of earth, so the video camera was needed to differentiate between safe and unsafe steering directions. However, in order to function outside of the densely vegetated areas, we needed the laser range camera for avoiding isolated mounds.

Once again our testing used both the monolithic neural network and the MAMMOTH network. The CCD FeNN from the MAMMOTH network was trained with freshly generated images of the new test region, the laser range FeNN was trained with the same exact training set used for the additive obstacle test, and the Task Net and monolithic network were trained with a set that was a concatenation of the training set for the CCD FeNN and an equal number of exemplars from areas of less dense vegetation.

In the few tests, the monolithic neural network was largely unsuccessful. The modular MAMMOTH network was able to go through the test region about 40% of the time. One cause of failure was training and testing on different days with different lighting. The other major cause was at the very slow speeds we had to use, the vehicle's front-mounted sensors would often clear an obstacle before the wheels did (and the current reactive-only system would think all was clear).

5. Conclusions

This work has demonstrated new capabilities in cross-country navigation and sensor fusion, and have continued an exploration of an alternate neural network paradigm, which produces more understandable and reusable neural networks.

For cross country navigation, the ability to handle vegetation intelligently was added. Training a network to avoid vegetation, work in simulation, and the preliminary field work suggest that we can also train the neural network to choose to drive over vegetation when the other options are worse.

The implications of this work for sensor fusion in general is that we can feel confident to add new sensor modalities to a given task. A monolithic neural network is capable of learning to fuse the sensing modalities, but the MAMMOTH architecture gives us even more control over how to use the data provided in any given modality. Using appropriate FeNNs we can add some of our a priori knowledge of useful features and at the

same time increase our ability to interpret how each sensing modality contributes to the task performance. Also, by tightly focussing the lower level internal representations of the network through the use of the FeNNs we are able to avoid cross-modality interference that can result in poor performance from keying on incidental features (such as the redness of a block in the simulated world).

Bibliography

[1] M. Daily, et al. Autonomous Cross Country Navigation with the ALV. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1988.

[2] I. L. Davis. *Neural Networks For Real-Time Terrain Typing*. Carnegie Mellon Tech Report CMU-RI-TR-95-06. 1995.

[3] M. Gluck and C. Myers. Hippocampal Mediation of Stimulus Representation: a Computational Theory. In *Hippocampus*. 1993.

[4] J. B. Hampshire and A. H. Waibel. The Meta-Pi Network: Building Distributed Knowledge Representations for Robust Multi-Source Pattern Recognition. IEEE PAMI Transactions. 1989.

[5] R. A. Jacobs, M. L. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*. 3:79-87. 1991.

[6] T. Jochem, D. Pomerleau, C. Thorpe. MANIAC: A Next Generation Neurally Based Autonomous Road Follower. *Proceedings of International Conference on Intelligent Autonomous Systems* (IAS-3). 1993.

[7] A. Kelly. *A Partial Analysis of the High Speed Cross Country Navigation Problem*. Carnegie-Mellon University Ph. D. Thesis Proposal. 1993.

[8] M. Marra, R. T. Dunlay, and D. Mathis. Terrain Classification Using Texture for the ALV. *Proceedings of the SPIE Conference on Mobile Robots*. 1992.

[9] L. Meeden, G. McGraw, and D. Blank. Emergent Control and Planning in an Autonomous Vehicle. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. 1993.

[10] D. Pomerleau, *Neural Network Perception for Mobile Robot Guidance*. Ph.D. Dissertation. Carnegie-Mellon University Technical Report CMU-CS-92-115. 1992.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Ed. MIT Press. 1986.

[12] F. J. Smieja and H. Mühlenbein. *Reflective Modular Neural Network Systems*. Technical Report, German National Research Centre for Computer Science. 1992.

[13] W. A. Wright. *Contextual Road Finding With A Neural Network*. Technical Report, Sowerby Research Centre, Advanced Information Processing Department, British Aerospace. 1989.

A Practical Stereo Vision System

1. Introduction

Stereo vision techniques offer a number of advantages to the designer of a robotic vehicle. Stereo relies on low-cost video technology that uses little power, is mechanically reliable, and emits no signature (unlike ladar). A stereo system also allows more flexibility; most of the work of producing a stereo range image is performed by software that can easily be adapted to a variety of situations.

To date, radar, sonar and single-camera vision have proven to be more popular than stereo vision for use on robotic vehicles. Two machines that have used stereo successfully are JPL's Robby and Nissan's PVS vehicle. The PVS system, however, does not need to produce a complete depth map [1], while Robby's stereo does not need to operate at very high speeds. The primary obstacle to stereo vision on fast vehicles is the time needed to compute a disparity image of sufficient resolution and reliability. With faster computers becoming available every year, performance is already much less of an issue.

A fast and robust stereo system was developed for use on autonomous vehicles: two robotic trucks, NAVLAB and NAVLAB II [2], and an 8-legged walker called Dante. To achieve a useful level of performance, we have been willing to trade resolution, image size and accuracy to gain speed and reliability. As higher performance computing becomes available for these vehicles, the dimensions and depth resolution of the stereo images will be increased.

2. System Design

Outdoor mobile robots typically share several requirements in a ranging system: reliable performance in an unstructured environment, high speed, and physical robustness. These requirements are crucial for two applications we envisioned when building our system: low-speed cross-country navigation and high-speed obstacle detection.

Obstacle detection is an important part of the system in order to drive our autonomous trucks at speeds of up to 55 MPH. The requirements for this task are that all major obstacles to movement be detected and that the system run quickly enough to allow time to stop the vehicle or avoid the obstacle. In many cases, the vision algorithms used to navigate the truck have no knowledge of the three-dimensional structure of the environment and cannot perform obstacle detection. Obstacle detection, when performed, used to be accomplished by a second process using sonar or laser range-finding. Since these sensors are short range, they are impractical for use at high speeds where long stopping distances are needed. Our stereo vision system can be tuned, through choice of lenses and camera separation, to detect obstacles at a variety of ranges, even out to 100 meters or more (which would be needed for obstacle detection at 55 mph).

Another application for stereo is to provide terrain information for a cross-country navigation system. In this instance, each range image generated by stereo is converted into a two-dimensional elevation map showing ground contours and obstacles. This map is then merged with previous maps and used by planning software to generate a path for the vehicle to follow through the terrain. Since the planner requires detailed information, the stereo range image must be accurate and of fairly high resolution. Fortunately, cross-country vehicles in rough terrain do not need to move at highway speeds and, since they work from a map, can plan moves ahead of time. This means that range images do not need to be generated as quickly for this application as for obstacle detection.

These two applications suggest a single system that can be tuned to produce images of increasing quality in an increasing amount of time. Estimated system requirements are detailed in a table below.

In both cases, the computation requirements are considerable and demand a simple, fast stereo technique. Our system, already working well at slower speeds on several vehicles, will meet these requirements and more. Our system is able to do this because of a number of important developments: trinocular stereo, the sum of sum of squared differences matching technique, and careful attention to detail in the design and implementation of the various system components. Each of these aspects of the system will be discussed below.

	OBS. AVOID	X-COUNTRY
Minimum range	3m	3m
Maximum range	50-100m	25m
Depth resolution	40cm @ 15m	20cm @ 15m
CPU time	0.1 sec	5 sec
Image size	256*120	512*240

Figure 1: Typical requirements for two stereo applications.

3. Trinocular Stereo

A three-camera (trinocular) stereo system was selected over the more usual two-camera model. The initial motivation for this choice was the hope that larger amounts of data would make the matching process easier. Moreover, the presence of the third camera was expected to help in the resolution of ambiguities when performing a match (consider the matching of two images of a repetitive pattern, such as a brick wall). Studies have shown that the benefits of a third camera outweigh the associated computational cost. Dhond and Aggarwal [3] found that the increase in computation due to a third camera was only 25% while the decrease in false matches was greater than 50%.

In our experience, the addition of the third camera produces a dramatic improvement in the quality of the range images obtained. Since our system requires the use of a long (1 meter) baseline, it may be that the third camera is important to bridging the dramatic disparities between the outer images when viewing objects close to the robot.

In a practical robot system, the trinocular approach has other advantages. When human help is far away, such as for our robot Dante, which will explore Antarctica, the third camera allows the robot to fall back on a two-camera system in the event that any of the three cameras fails. Finally, the trinocular approach makes good use of the typical 3-band video digitizer, which can support one monochrome camera on each of the red, green and blue channels.

Figure 2: The Dante Robot.

4. The SSSD Method

Once a set of three images has been taken, they must be converted to a range image. The fundamental principle behind this type of stereo is that when the same scene is imaged by more than one camera, objects in the scene are shifted between camera images by an amount that is inversely proportional to their distance from the cameras. To find out the distance to every point in a scene, each point in one image must be matched with corresponding points in the other images. Many methods have been used to perform this matching (many successful), including feature-based matching, multi-resolution matching, and even analog hardware-based matching. The approach to performing this match is to use an SSSD (sum of sum of squared differences) window. This technique, developed by Okutomi and Kanade [4] has proven to have many advantages.

The SSSD method is simple and produces good results. The technique also places no limitation on the scope of the stereo match. This allows production of small, low resolution images to be performed as easily as production of larger, high resolution images. Even more importantly, the technique easily allows the incorporation of our third camera. Because of its regularity, the SSSD method is easily adaptable to both MIMD and SIMD parallel machines. Lastly, as shown below, the SSSD makes it easy to compute a confidence measure for each pixel in the range image.

The SSSD method is used to determine which pixels match each other between our input images. When looking for matching pixels, we have several clues to help us. The first is that, due to the geometry of our cameras, which are arranged in a line, matching pixels will occur on the same scanline in each image. Due to the baseline of the cameras, the disparity (horizontal displacement of a pixel) must fall within a certain range. For each pixel in the first (right-hand, in our case) image, we need, then, to look at a small range of pixels on a single scanline in each of the other images. The pixel in this range that produces the best match is considered to be same point in the real scene. Once we have this match, we can then immediately calculate the range to that point in the scene.

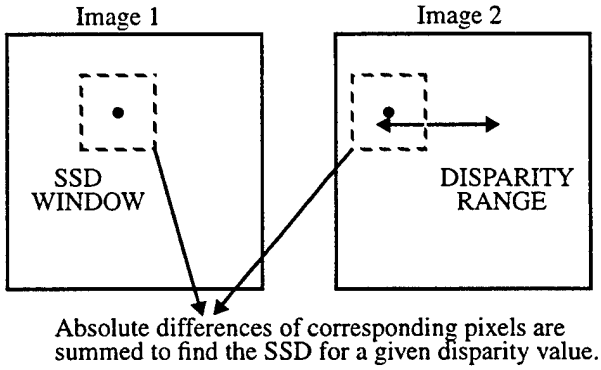


Figure 3: Computation of the SSD (2-camera case).

The trick, of course, is to figure out which in the range of possible pixels is the right match. For two images, the SSD method works by comparing a small window around the pixel in the original image to a window around each of the candidate pixels in the other image. The windows are compared by summing the absolute (or squared) differences between the corresponding pixels in each window. This yields a score for each pixel in the range. The pixel with the lowest score has a window around it which differs the least from the window around the original pixel in the right-hand image.

The sum of sum of squared differences (SSSD) is simply the extension of the SSD technique to three or more images. In our case, we have three camera images; for each pixel we perform an SSD match between the right-hand image and the center image as well as between the right-hand and left-hand images. For each disparity "D", we must examine the window shifted by D pixels in the left-hand image and by only D/2 pixels in the

center image. When the SSD of both pairs of windows has been computed, the two SSD values are summed to produce a single score (the SSSD) for that disparity value.

The size of the SSSD window is an important parameter in the stereo computation. A larger window has the effect of smoothing over small errors in the stereo matching, while also smoothing away many of the details and smaller features in the image. We typically use as small a window as we can that will still produce a fairly error-free range image (typically, 10 rows by 20 columns). The SSSD window does not have to be square, and we find for our applications that it is better to widen the window, sacrificing horizontal resolution, than to increase the height at the expense of vertical resolution.

In Okutomi and Kanade's original SSSD system, variable window sizes for each pixel in the image were used to achieve the best results for each portion of the image. Also, disparities were sampled at the sub-pixel level (with interpolation of image pixels) to increase depth resolution. These enhancements, while giving superior results, are too slow for our application so they are not used.

We used a number of techniques to speed up our computation. Due to our wide camera baseline, we typically have a disparity range of 120 pixels to search. Instead of checking for sub-pixel disparity, however, we do the opposite. The wide baseline of the jig gives us acceptable resolution at longer ranges, but it gives us much more resolution than we need at short ranges (2cm resolution at 3m range). To speed things up, many disparities may be skipped at the shorter ranges, while checking the full range of disparities at longer ranges. This has the effect of equalizing our resolution over the range of the system, while reducing the number of disparities calculated to about 50.

When performing the SSSD, we have improved performance by reversing the order of the computation. Instead of finding the SSD between two sets of windows and then summing these values, we first compute the differences between the whole images and sum them to produce a single image representing the match at that disparity. The window around each pixel is then summed to produce the SSSD for that pixel. The summation of windows can be done very quickly because we maintain rolling sums of columns to speed the computation.

Another technique we use to speed up computation is to reduce the sizes of the input images. For typical cross-country work, the full vertical resolution is not necessary, so we use images of 512 columns by 240 rows. For obstacle avoidance, a smaller image of 256 by 120 pixels will suffice because small details in the scene are not important for this application.

For all the compromises made in the interests of speed, the range images produced by this system are surprisingly clean. Sometimes, however, the SSSD technique will break down when there is not enough texture in the image to perform a good match.

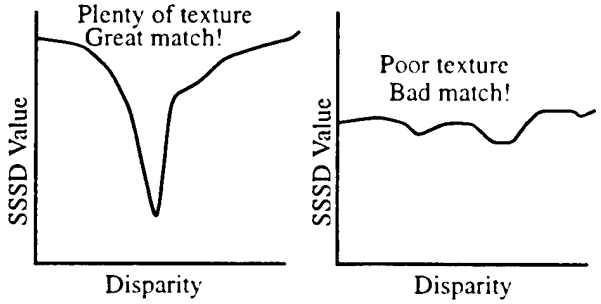


Figure 4: Two examples of SSSD curves.

For example, an image of a smooth, white wall will produce the same SSSD score for every disparity; a graph of the SSSD values will look like a flat line. When there is plenty of texture, there is almost always a clear minimum SSSD value on the curve.

To make use of this phenomenon, a "confidence" value is computed for each pixel in the range image. This is a measure of the flatness of the SSSD curve. If a pixel in the range image has a confidence level below a predefined threshold, it can be ignored as unreliable. The confidence value for each pixel is computed by taking the average of the percent of change between successive SSSD values. For a given pixel, the confidence value C can be expressed as a function of the SSSD values for that pixel, S(d) for the range of computed disparities dmin through dmax:

$$C = \left(\sum_{d = d_{min}+1}^{d_{max}} \frac{MAX(S(d), S(d-1))}{MIN(S(d), S(d-1))} - 1 \right) / d_{max} - d_{min}$$

5. Hardware Details

The development of several pieces of special hardware turned out to be critical to the success of our stereo system. The most complex item was the jig used to hold our three cameras. The SSSD algorithm requires that the stereo cameras be displaced by equal amounts along a straight baseline. Each camera is pointed in the direction precisely perpendicular to the baseline, and the roll and elevation of the cameras are adjusted to cause the scanlines of each camera to be vertically coincident. Experiments have showed that this camera alignment must be quite precise if the system is to work well. While misalignment could perhaps be corrected in software, in the interests of speed it was decided to build a mechanical fixture that would guarantee alignment.

Unfortunately, we have found that typical CCD cameras and lenses exhibit considerable differences in image alignment with respect to the camera body. The cameras could not be bolted into a precisely machined stand. Instead, an adjustable mount was needed for two of the cameras which allows them to be carefully aligned with the third camera.

The camera fixture, or "jig", consists of a rigid bar, 1 meter long, with mounting points for three cameras. The center camera is mounted to a fixed platform, while the left and right cameras are attached to adjustable platforms.

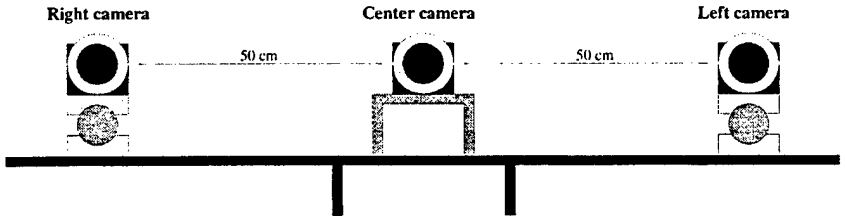


Figure 5: Front view of stereo camera jig.

The adjustable platforms have three rotational degrees of freedom and are designed to allow minute adjustments to be made in the orientation of the cameras. The platforms may also be rigidly fixed in place with locking screws to keep them well aligned during rough handling.

The choice of baseline (distance between cameras) is critical. With a very short baseline, there is little change between the three camera images. This translates to poor depth resolution. On the other hand, longer baselines will have the effect of decreasing image overlap (and thus, the effective field of view) and complicating the process of finding stereo matches. The choice of a 1-meter baseline was a trade-off between these two concerns and was intended to give us good depth resolution without ruining our field of view. Due to this choice, depth resolution at 15 meters is not as good as hoped; however, at closer ranges resolution is still very good.

The cameras used are small Sony XC-75 monochrome CCD cameras. These cameras were found to be more mechanically sturdy than average. Our previous cameras had a slightly flexible mounting system for the CCD

element, which would slip out of alignment on the first bump. The Sony cameras were modified by adding a stiffening/mounting plate to the bottom. This plate serves to stiffen the camera body as well as to provide a better mount point than the standard single-bolt tripod mount. Another advantage of the XC-75 is the electronic shutter system, which serves to prevent motion blur in the images.

Auto-iris lenses are a must for outdoor work. Autoiris lenses with a focal length of 8mm were selected to give a moderately good field of view. Although 6mm lenses would have produced a greater field of view, they introduced enough distortion to significantly degrade the quality of our results. Since we did not want to use CPU time to unwarp each image, we chose to substitute 8mm lenses.

As in the case of the cameras, some modifications to the lenses were necessary. We were unable to find any lenses that were mechanically sturdy enough to resist the vibration and bumps of our vehicles. The average lens is comprised of three important assemblies: the lens elements, a focus mechanism and a camera mount. The several sets of threads that comprise the focus mechanism are typically very sloppy, and since they remain mechanically unloaded, they allow the lens elements to move relative to the camera mount. The movement of the lens elements causes a shift in the image falling on the CCD. In some lenses, a light tap on the lens body was enough to put the image out of alignment by as much as 10 pixels. Our solution to this problem was to discard all but the lens elements. We fashioned a new, single piece aluminum adapter between the lens elements and the Sony camera which allows no movement between the two. Of course, this also had the advantage of permanently fixing the focus which is a benefit on a moving vehicle.

The digitizer used to capture the video images for processing was a conventional 24-bit color framegrabber. The 3 monochrome cameras were synced to each other and connected to the red, green and blue channels of the digitizer. The video gain feature on the digitizer was found to be useful to balance the gains between the three cameras. We found that if the images were not close enough in gain-levels, our results were badly affected.

6. Results

Figure 6: Original right-hand image.

Results obtained to date with this system have been very encouraging. We have successfully driven our HMMWV truck through a field of obstacles at a nearby slag heap under autonomous control. The system has also guided our 8-legged robot (Dante) during outdoor runs.

An example input image is shown in Figure 6. This image is one of a set of three images taken with the stereo jig. The image shows a mostly flat piece of terrain containing a traffic cone. Figure 7 shows the computed range (disparity) image. The light-colored portions of this image represent areas closer to the camera, while darker pixels are more distant. The traffic cone appears as an area of medium-gray in the center left of the image. The image contains a number of errors including the blobs floating at the top of the image and the fall-off at the right side. These errors are easily removed during post-processing when we generate an elevation map. The elevation map, which shows the terrain from above as in a topographical map, is the map most commonly used to plan routes for our robots. The elevation map generated from this range image is shown in Figure 8.

The elevation map is a view of the terrain seen from above. In this map, the robot is situated at the center of the left side of the image and is facing towards the right. Lighter areas in the image represent higher elevations while darker shades of grey represent depressions. The black border represents the area outside the field-of-view of the sensor. The traffic cone can be seen as the vertical white line near the center of the image.

The stereo system has been implemented on several conventional workstations as well as on a number of parallel machines, including an iWarp, a 5-cell i860 and a 4096 processor Maspar machine. Our algorithms have proven to map well to parallel machines, and as can be seen in the table shown in Figure 9, this has led to dramatic improvements in performance. The times for the iWarp are given because this machine is used on our NAVLAB vehicles.

Figure 7: Computed range image.

This stereo system has been tested in a wide range of environments including a slag heap, a variety of roads, indoors, and in wooded areas. The results have been very good in almost every situation. The only failing of the system is that it does not perform well in areas that exhibit few features and little surface texture. These areas have included sky, newly paved roads and fresh snow cover (our Antarctic mission is in a mostly snow-free area). Even a small addition of texture to a scene, such as tracks in the snow or leaves on a new road, has proven to be enough to overcome this problem. The confidence measure, which is essentially a measure of surface texture, does a good job of detecting areas that are too smooth to yield good matches. This means that, while you may not get any useful data, the robot will not be misled by low-texture errors in the range image.

Other errors in the range image are inevitable due to the occasional presence of areas in the input image that fool the SSSD matching process. Typically, we have found that these errors are both small enough to ignore, and also produce features that are easily filtered out. For example, if a small object seems to be floating in the

air in front of the robot, it is ignored.

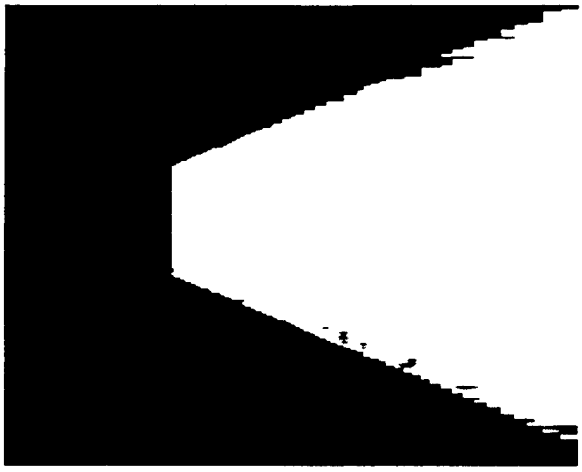


Figure 8: Computed elevation map.

MACHINE	IMAGES	DISPARITIES	TIME
Sun Sparc II	256*240	16	2.46 sec
16 Cell iWarp	256*240	16	0.35 sec
64 Cell iWarp	256*240	16	0.15 sec
Sun Sparc II	512*240	27	8.03 sec
Sun Sparc II	512*480	88	56.34 sec
64 Cell iWarp	512*480	88	2.18 sec

Figure 9: Times for a variety of stereo computations.

7. Conclusion

A stereo vision system was developed that has proven itself through application to several real-world robots. The keys to the success of this system were a simple, straightforward approach to the software and attention to hardware details. This system has made it clear that stereo is a practical, competitive alternative to other perception systems in use on mobile robots.

Future work with this system will concentrate on two areas: increasing the speed of the system, and improving the quality of the images. Speed improvement is expected to be possible through further parallelization of the algorithm as well as the use of faster hardware. Improvements in the algorithm may include the use of variable window sizes and sub-pixel disparity checking.

Bibliography

[1] T, Ozaki, M. Ohzora, and K. Kurahashi. An Image Processing System for Autonomous Vehicle. In

Proceedings SPIE Vol. 1195 Mobile Robots IV. 1989.

[2] C.E. Thorpe (editor). *Vision and Autonomous Navigation, The Carnegie Mellon Navlab.* Kluwer Academic Publishers. 1990.

[3] U.R. Dhond and J.K. Aggarwal. A Cost-Benefit Analysis of a Third Camera for Stereo Correspondence. *International Journal of Computer Vision.* 6:1, pp. 39-58. 1992.

[4] M. Okutomi and T. Kanade. A Multiple-Baseline Stereo. In *Proceedings CVPR.* 1991.

Sonar-Based Outdoor Vehicle Navigation

1. Introduction

Compared to light-based sensors, sonars have the advantage that they do not get confused by transparent or black surfaces. On the other hand, the wavelength of ultrasound is much larger than the wavelength of light, i.e. usually around 4 mm as compared to 550 nm for visible light. Therefore, unless the transducer faces the reflector surface in a normal direction, only rough surfaces or edges can reflect sound waves. However, most real world outdoor surfaces almost always have a type of surface roughness that enables a sonar to detect the object.

Sonar sensors were used in one of the collision avoidance systems of the Navlab. The following sections describe the sonar system and its performance in an outdoor environment. Some novel results were obtained in using the system for vehicle navigation by itself and by integrating it into other vehicle navigation systems. The system is configured in such a way that more sensors can be added easily in the future. These sensors do not necessarily have to be sonars but can be any other type of point range sensor. In the future at least one other type of point range sensor into the system, most probably laser or radar based will be integrated. For example, the ERIM laser range finder has already been successfully integrated as part of a cross-country navigation system. In this case, however, obstacles can only be detected by first processing 3-D range images [6].

2. Hardware Configuration

Sonar sensors have already been used successfully for indoor mobile robots as described in [2], [3], [9] and [7]. An outdoor environment, however, adds some additional constraints on the type of sensor that can be used. Specifically the sensor should meet the following requirements:

- Robust against moisture and dust particles.
- Robust against noise from vehicle engine and other sound sources.

Therefore an open type electrostatic transducer such as the Polaroid cannot be used. Instead a closed type piezoceramic transducer operating at a frequency of 80 kHz was selected. A detailed description of this sensor is given in [8]. The high operating frequency makes this sensor fairly robust against acoustic noise, while still providing an operating range up to 6 m. The beam angle of the sensor is approximately 5°, i.e. at the 3 dB level, intensity falls off from the major axis. Based on these characteristics, a total of five sensors was chosen in order to provide a good area coverage in front of the vehicle with a reasonable spatial resolution. The sensors are mounted on a guide rail such that their position and orientation can be freely adjusted. A typical sensor arrangement is shown in Figure 1. Certain sensor configurations or environments can lead to acoustic interference between individual sensors. Therefore the hardware provides the ability to choose the exact trigger time of each sensor. In most circumstances the sensors are mounted such that they point away from each other. In this case all sensors are triggered at the same time. At present, a measurement rate of 9 Hz is used, which is based on the following calculations. For very good reflectors we can assume a maximum operating range of 8 m, which corresponds to a time of flight of sound in air of approximately 50 ms. Thus, echoes are considered during a receiving period of T_{rec} = 50 ms after triggering the sensor. In order to avoid measurement errors due to multiple echoes, only the range of the first echo is measured. The sensors are retriggered after an additional wait period of T_{wait} = 60 ms, which ensures that all detectable echoes from previous pulses are attenuated below the detection threshold. Thus, the total cycle time $T = T_{rec} + T_{wait}$ = 110 ms.

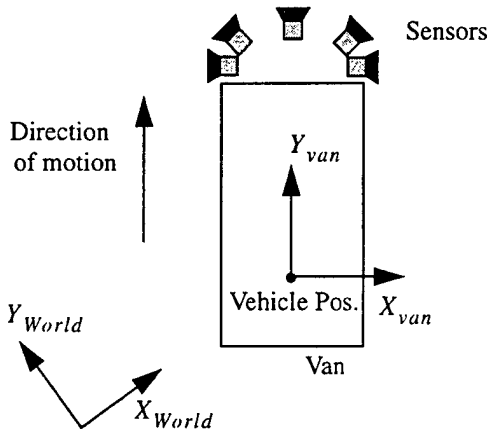


Figure 1: Sensor configuration.

Each sensor measurement is tagged with the position of the vehicle. At present, the Navlab uses dead reckoning to estimate its position relative to some initial point. The distance traveled is provided by an optical encoder on the drive shaft and vehicle orientation in 3-D space is provided by the gyroscope of an inertial navigation system. The measurements are combined to give the x-y position and orientation of the vehicle with respect to world coordinates. For the position tag of a sonar measurement, only the following three parameters are used: x, y and φ, where φ = yaw (Figure 1).

The hardware of the sonar system consists of an interface module, which triggers the sensors and measures the time of flight to the first echo returned. The interface module is accessed via VME bus from a 68020-based CPU. This processor runs as a slave of the vehicle controller processor under a real-time operating system and takes care of data acquisition, conversion to range, position tagging and proper timings. The map building and tracking algorithms are presently implemented on a Sun SPARC station, which communicates with the vehicle controller via ethernet. Ethernet communication time uncertainties can be neglected because of the comparatively long cycle time of the sonar system. The hardware configuration is shown in Figure 3.

3. Local Grid Map

This sensor system can now be used to build a local grid map. The grid map is local because it contains only information about the immediate surrounding of the vehicle. The vehicle position is kept at a fixed point in the map. As the vehicle moves, objects in the map are moved from cell to cell relative to vehicle position. Once an object falls outside the map boundary, it is discarded and the information is lost. Using just a local map has the advantage that error accumulation owing to dead reckoning is kept small, since only relative movements are considered. On the other hand, the disadvantage is that information is lost and thus no global information is available. However, if desired, sequences of the output from the local map could be combined and included in a larger global map. At present, the area covered by the local map is from –8.2 m to 8.2 m along the x-axis and from –22.2 m to 70.2 m along the y-axis. Each grid cell has a resolution of 0.4 m along the x-axis, and a variable resolution of 0.4 m , 2 m or 4 m along the y-axis, depending upon distance from the vehicle. Hence the map consists of 41 x 101 cells (See Figure 3). The reason for taking such a coarse resolution for each cell is that most applications of the system do not require a high accuracy, and the size of the highest resolution grid cell is small with respect to the size of the vehicle. As objects get further away from the vehicle, a coarser map resolution is adequate. In general, sensor readings become less accurate as the object is further away. However, coarse resolution map information is sufficient to plan initial navigation maneuvers. As the object moves closer, navigation needs to be more precise and therefore map resolution is increased.

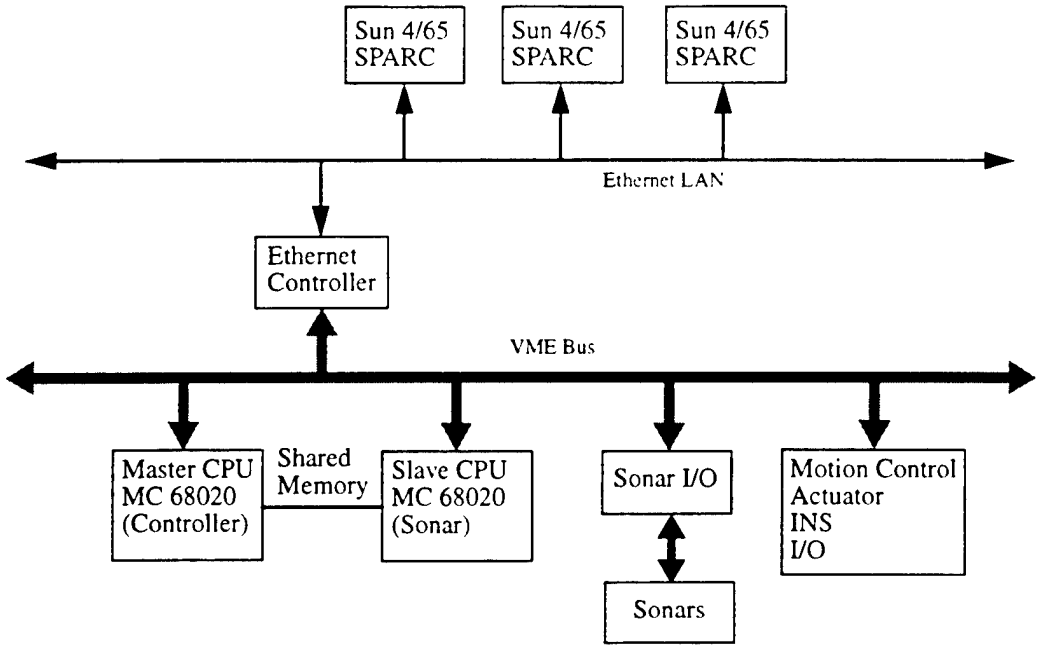


Figure 2: Hardware architecture.

3.1. Grid representation

This grid map representation can be used with any obstacle detection sensor, such as sonar, laser (single spot or scanned), radar or stereo. Currently, only an array of sonar sensors is used as described in section 2. This sensor does not always provide an accurate measurement. For the particular sonar sensors chosen for example, the measurement accuracy is about ± 1 cm. However, depending on the environment, the sensor may also deliver noisy results. The reason lies in the poor angular resolution of the sensor. Echoes may be detected from an object at a far range in the main lobe or from a good reflector at closer range in a side lobe. Depending on the relative signal strengths and movement of the sensor, the range reading may oscillate. A similar effect can also happen when an echo that is reflected multiple times is received. The sonars have a fairly small maximum range. However, in anticipation of future additions of longer range sensors based on radar or laser, the maximum forward coverage of the map is large at multiple resolutions.

Each cell has a set of parameters or annotations associated with it, which are described below:

1. **Object Type**
   This parameter is used to indicate if the object in that cell was seen at the current sensor reading, or if it was seen at a previous reading. If it was seen only at a previous reading, then the object type indicates that it must have moved to that particular cell due to vehicle motion only.
   The parameter is also used to denote which type of sensor detected that particular object if several different types of sensors are connected (e.g. sonar or radar).

2. **Position**
   Indicates the x-y position of the object with respect to vehicle position.

3. **History**
   This parameter counts the number of times an object was detected in a particular cell.

4. **Curvature Vector**

Is precomputed for each cell and denotes the range of steering arcs that would avoid a collision between the vehicle and an object in that cell.

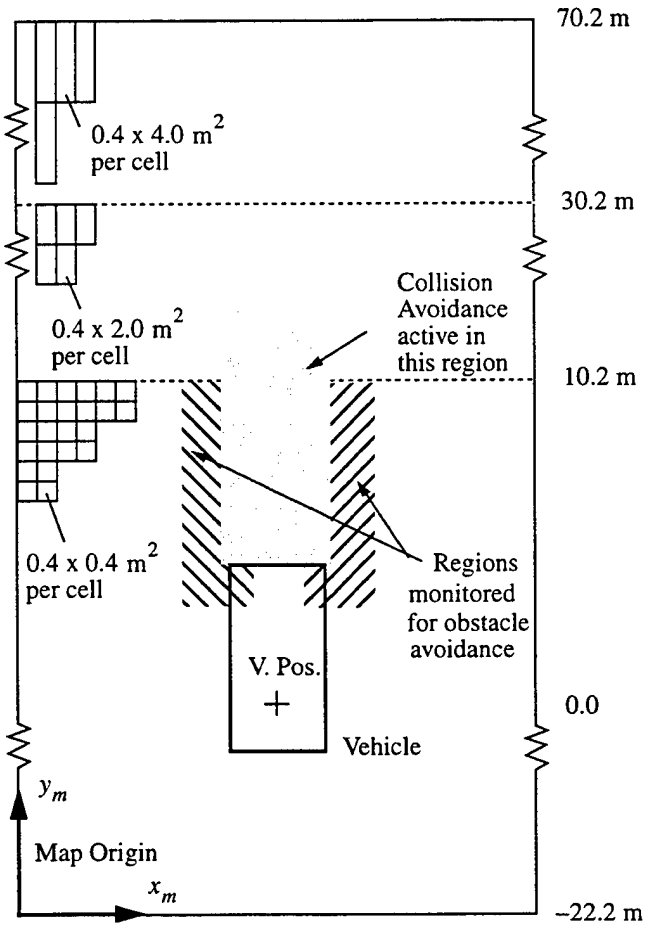


Figure 3: Local grid map.

The resolution of the grid is fairly coarse and hence a position parameter (X_{obj}, Y_{obj}) is kept to avoid gross error accumulation when objects are transformed in the map. Only one object is kept per grid cell.

Measurement uncertainty is part of the grid cell representation and any object detected within an area covered by a particular cell is taken to belong to the same object.

3.2. Grid Transformation

A short description of object transformation within the map follows: Vehicle position and orientation are kept constant within the map. Therefore objects in the map move with respect to the vehicle. The vehicle's positioning system returns vehicle position and orientation with respect to a global frame of reference (x, y) that is determined at the time of initialization of the system. Since we are interested only in the movement of objects with respect to the vehicle's coordinate system (x_m, y_m), the appropriate transformation is obtained by using the position increments along the axis $(\delta x, \delta y)$ and orientation $\delta\varphi$. The total distance traveled is given by $\sum \delta s$. As the map remains local, errors due to dead-reckoning are reduced and the positioning system can be

initialized independently by using this method. Hence, if the vehicle moves from a position in a plane (x_1, y_1, φ_1) to a new position (x_2, y_2, φ_2), then an object in the map at position (x_{m1}, y_{m1}) is transformed to position (x_{m2}, y_{m2}) as follows (Figure 4):

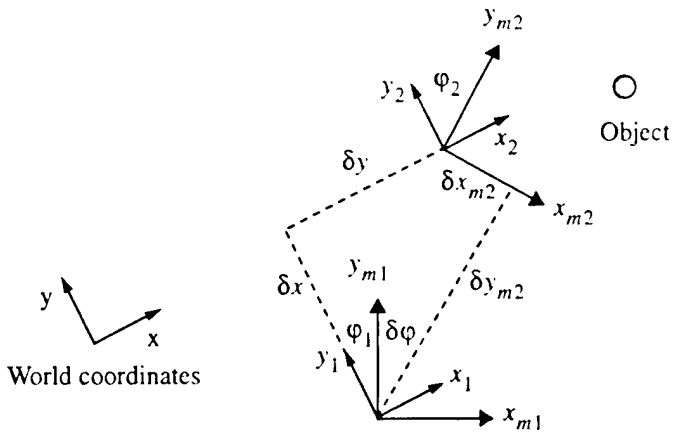


Figure 4: Object transformation in local grid map.

Position and orientation increment:

$$\delta x = x_2 - x_1\ ,\ \delta y = y_2 - y_1\ ,\ \delta\varphi = \varphi_2 - \varphi_1\ ,\qquad \delta s = \sqrt{\delta x^2 + \delta y^2}$$

$$\delta x_{m2} = \delta x \cos\varphi_2 + \delta y \sin\varphi_2 \qquad \delta y_{m2} = -\delta x \sin\varphi_2 + \delta y \cos\varphi_2 \tag{1}$$

New object position (vehicle moving forward):

$$x_{m2} = x_{m1}\cos\delta\varphi - y_{m1}\sin\delta\varphi - \delta x_{m2}$$

$$y_{m2} = x_{m1}\sin\delta\varphi + y_{m1}\cos\delta\varphi - \delta y_{m2} \tag{2}$$

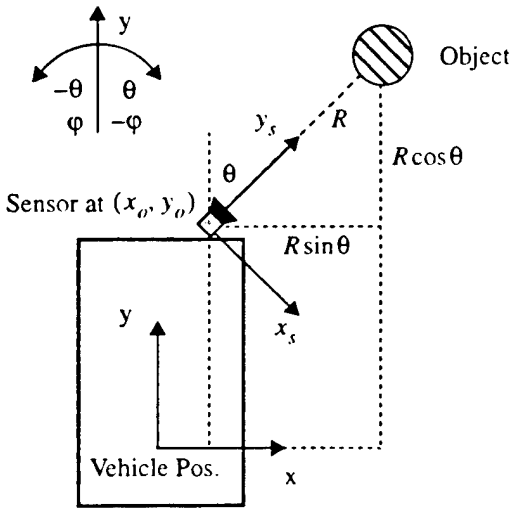


Figure 5: Sensor to object transformation.

After the position of objects in the map is updated, new objects detected by the sensors are added. A sensor

measures the range R to an object. Position and orientation of each sensor on the vehicle are known. Hence, using the transformation "sensor position $\rightarrow$ vehicle position $\rightarrow$ map", a new object is placed in a cell in the map (Figure 5). If that particular cell is already occupied, then only the cell parameter 'History' is updated as described in 3.3., otherwise all cell parameters are updated.

3.3. Filtering

The map parameter **History** is used for differentiating between moving and stationary objects and for filtering the data. **History** can also be used to evaluate the confidence that a particular cell is occupied by an object. A higher value of **History** indicates a higher confidence. In the case of collision avoidance, for example, it is desirable to slow down the vehicle if there is an obstacle in front and to resume driving if the obstacle moves away, like a car or person. Hence, objects in the map in a sensor's field of view are deleted if a sensor does not detect them anymore. However, an object will not be deleted immediately, but only after it was not seen by the sensor for a certain time period. This time period is defined by a parameter called **Life Time**, which is given as the number of cycles of the update sequence (Figure 7). We can assume that the cycle period is approximately constant. The parameter **History** is therefore updated at time t as follows:

- If an object is detected, then

  1. If cell is empty, History_t = Life Time
  2. Otherwise, $\text{History}_t = \text{History}_{t-1} + 1$
  3. Decay Amplitude = 0

- If no object is detected by the sensor and History is not equal to zero, then

  1. Calculate Decay Amplitude initially, i.e. if it is zero:

     Decay Amplitude = History_t / Life Time

     Decay Amplitude is calculated only once at the beginning of a decay sequence. It ensures that each object disappears after the same amount of time, i.e. Life Time, has passed.

  2. $\text{History}_t = \text{History}_{t-1}$ – Decay Amplitude

Since different types of sensors may have different fields of view, a field of view constraint is applied depending on which type of sensor detected a particular object. Each map grid cell is labeled as to whether it is within the field of view of a particular sensor. In the case of sonar sensors, it is not very useful to consider the field of view of side looking sensors. Because of vehicle speed and driving direction, objects detected in these areas are seen only for a short time and it would not be possible to apply a proper decay. Therefore, the decay algorithm is applied only to objects appearing in the area directly in front of the vehicle. Moving objects appearing in this area are also of most concern for safe vehicle navigation. Therefore the velocity of the vehicle is reduced as a function of range to the closest object detected in this area. The velocity is set to zero if the closest range is less than a certain minimum safety distance. The decay algorithm ensures that the vehicle resumes driving when the obstacle moves away. Figure 6 shows a plot of the percentage of vehicle velocity set versus closest range. D_{max} corresponds to the maximum sensor operating range. The gain a is calculated as

$$a = v_{max}^2/(D_{max} - D_s) \tag{3}$$

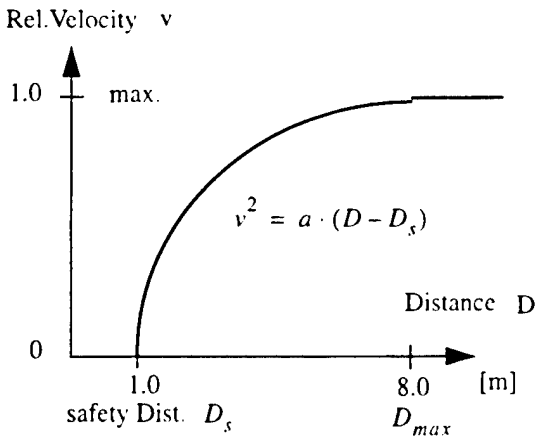


Figure 6: Velocity control.

The parameter **History** can also be used to eliminate spurious echoes, such as the ones returned by rain droplets. In this case an object is supposed to be actually present only if it has been seen consecutively for a certain number of cycle times. In practice, a threshold of Life Time + 2 has worked well. Since rain droplets return echoes at random ranges as time progresses, these returns can then be filtered out and will not appear as ghost objects in the map.

Figure 7 shows the cycle for map operations:

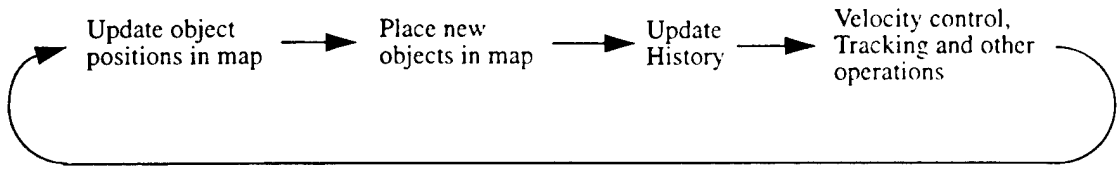


Figure 7: Updating sequence for local grid map.

The parameter **Object Type** is used to denote the sensor type that detected the object, i.e. for example sonar or laser if these sensor types are connected. In addition, it also indicates the type of operation that was performed on an object in a particular grid cell. These operations can be transformations as mentioned before, or filtering operations like the ones described in the following sections. **Object Type** is also used to tag certain objects for tracking as described in section 6..

At present, each grid cell is associated with only four parameters. The map structure allows an extension to other annotations in the future, such as object characteristics or triggers. In this way the local grid map could easily be integrated into an *Annotated Map* such as the one described in [11].

Using a histogram-based method provides fast map building for real-time outdoor mobile robot navigation. This has also been shown in a similar approach for sonars in [2]. Probability-based methods, such as described in [9] and [3], provide more accurate and detailed map data. This is achieved by using many data samples and a high resolution grid. However, it is computationally more expensive and therefore not as well suited for fast navigation. If only few data samples are available as in our case, the performance of a probability-based method will degrade correspondingly and a histogram method will be more efficient.

4. Obstacle Avoidance

The data collected by the local grid map can now be used for autonomous vehicle navigation functions. A basic navigation function is the avoidance of obstacles in the environment of the vehicle.

The problem here consists of finding a range of arcs that the vehicle can safely travel on without colliding with an obstacle. In order to represent arcs, curvature is used instead of radius since curvature space is continuous, whereas radius space is not. For each cell in the grid map as described in section 3., a range of arcs can be determined that would make the vehicle collide with an object in that cell. Computing this range of arcs for each grid cell and merging the results from the entire map will determine the final range of arcs that the vehicle can safely drive on and those that would lead to a collision. In order to simplify the merging of results from each individual grid cell, the total range of steering arcs is linearly discretized into a set of 31 arcs, i.e from maximum curvature left to maximum curvature right. The number of arcs was chosen to be 31 as it can be easily represented as a 32-bit integer number for efficient computations. Figure 8 shows the set of discretized arcs that are evaluated for collision avoidance.

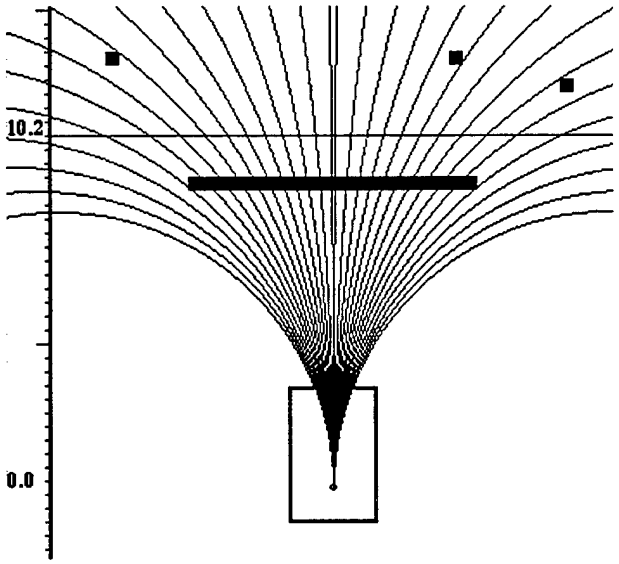


Figure 8: Arc evaluation for collision avoidance.

The range of inhibited arcs for each grid cell can be precalculated as follows. The center of the obstacle is taken to be at the mid-point of the cell. Since it is undesirable to pass an obstacle too closely, the obstacle is assumed to fill the entire grid cell and in addition is expanded by a safety margin M_s. A particular expanded object blocks not just any arc that it touches, but often a range of nearby arcs as well, due to vehicle size. We therefore calculate the boundaries of the range of arcs that would make the vehicle collide with the object. If the expanded object is inside the arc, the rear inside wheel tracks the innermost path and this defines the widest part that needs to be checked for collision (Figure 9(a)). If the object is outside the arc, the outside front corner swings widest and sets the collision limit (Figure 9(b)).

Hence, in the first case the left-bound radius of the inhibited steering arcs R_{Lb} is given by the arc OP with respect to Object 1 as shown in Figure 9(a). We therefore use the geometry shown for Object 1 and obtain,

$$\tilde{R}_{Lb} = (x_L^2 + y_L^2)/(2x_L) \qquad x_L = x_{obj} - M_s \qquad y_L = y_{obj} + M_s$$
$$\therefore R_{Lb} = \tilde{R}_{Lb} + 0.5V_w \qquad Curv_{Lb} = 1/R_{Lb} \tag{4}$$

where (x_{obj}, y_{obj}) are the coordinates of the centre of the object and V_w is the width of the vehicle.

In the second case the outside front corner point will hit an obstacle first in a collision and the geometry shown for Object 2 in Figure 9(b) is used. $\tilde{R}_{Lb}$ is then given by the circle that passes through points C and D with its centre $(-x_o, 0)$ on the x-axis. If the distance from vehicle position to the front of the vehicle is V_F, then the coordinates of point C are $(0.5V_w, V_F)$. The coordinates of point D are $x_D = x_{obj} - M_s$ and $y_D = y_{obj} - M_s$. Using the equation of the circle, we then obtain,

$$\tilde{R}_{Lb} = \sqrt{(x_D + x_o)^2 + y_D^2} = \sqrt{(x_C + x_o)^2 + y_C^2} \qquad \Rightarrow \qquad x_o = \frac{x_D^2 - (0.5V_w)^2 + y_D^2 - V_F^2}{2(0.5V_w - x_D)} \tag{5}$$

The left-bound radius of the inhibited steering arcs R_{Lb} is then given by,

$$R_{Lb} = -\left(\sqrt{\tilde{R}_{Lb}^2 - V_F^2} - 0.5V_w\right) \tag{6}$$

Note that in the above equation, R_{Lb} is negative for the given geometry as it indicates a left turn. Radius is then again converted to curvature.

In a similar way the right-bound radius (curvature) of the inhibited steering arcs R_{Rb} $(Curv_{Rb})$ can be calculated (See Figure 9 (b)).

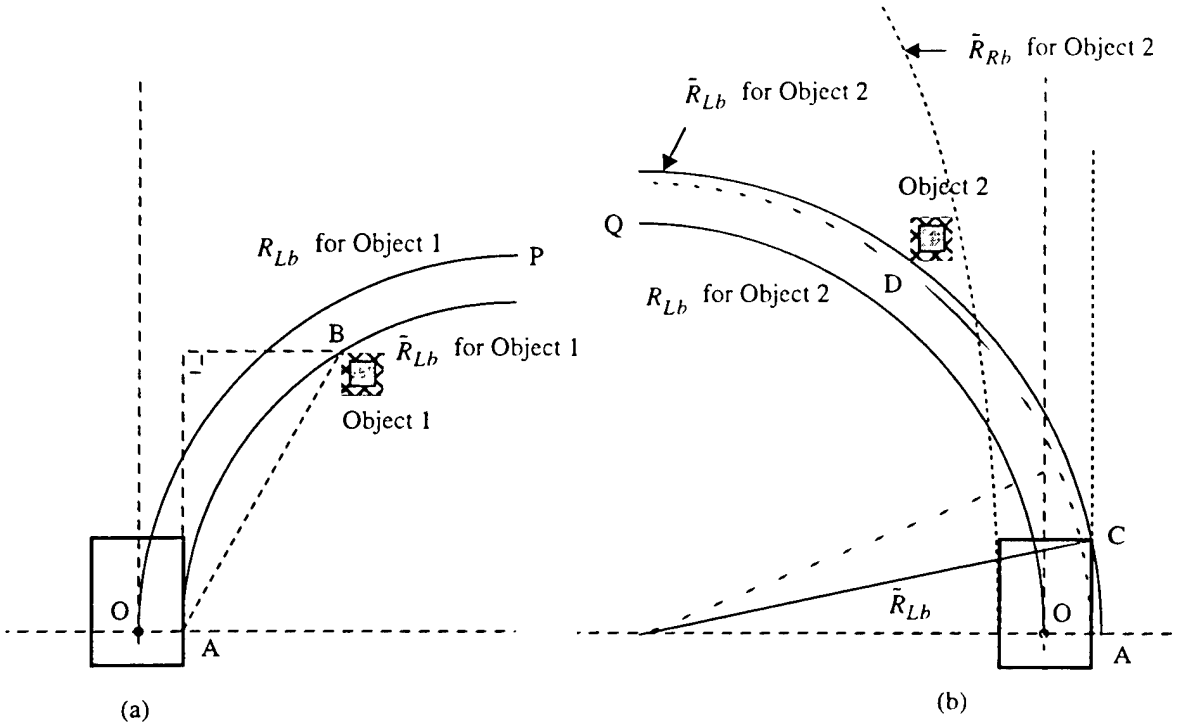


Figure 9: Precalculating arcs.

For each grid cell we now have the bounds on the inhibited steering arcs $Curv_{Lb}$ and $Curv_{Rb}$. The set of 31 discrete steering arcs can be represented using a 32-bit number or bitvector. Each bit represents a steering arc and a value of 1 indicates permission to drive on the respective arc; a bit value of 0 indicates that the respective arc is inhibited. Bit 0 (LSB) is always ignored here since only 31 bits are needed. Given the bounds on the inhibited steering arcs, the bitvector representing permitted and inhibited arcs for each grid cell can be calculated and stored. An example of a bitvector is given in equation (7). Here the zero curvature direction (straight ahead) and the three extreme left curvatures are inhibited. All other directions are permitted.

$$\vec{P} = 00011111111111101111111111111111 \quad (7)$$

During actual program runtime, only the bitvectors for each obstacle cell need to be ANDed and the resultant bitvector indicates permitted and inhibited arcs for all obstacles in the vehicles environment, i.e.

- Initially: $\vec{P}_{Result} = 11111111111111111111111111111111$
- For each cell *i*: $\vec{P}_{Result} = \vec{P}_{Result}$ AND $\vec{P}_i$

If the vehicle moves through an obstacle field along a desired direction, such as one given by a feature tracker (see Section 5.), an unobstructed direction closest to the desired direction is selected from the resultant bitvector. In certain circumstances this procedure can lead to oscillations in the vehicle steering, especially if the spatial distribution of obstacles changes rapidly or when approaching a wall (Figure 8). To prevent this from happening, a mechanism is used that keeps the vehicle turning in one direction, either left or right, once a direction has been selected in order to avoid an obstacle. The mechanism resets when no more obstacles are present or if the difference between desired steering direction and the new selected steering direction is less than 10% of the number of discrete arcs, i.e. three arcs here. The condition for the latter case indicates that there is little danger of oscillations as desired and selected steering direction are very close.

In addition, the vehicle is prevented from switching from extreme left turns to extreme right turns and vice versa unless an intermediate turn is selected in between.

These mechanisms were implemented as a result of observations of the behavior of the algorithm in simulation and in practice on the real vehicle.

5. Object Tracking

Another basic navigation function is the tracking of features in the environment and using this information to determine a path that the vehicle can drive. The following paragraphs describe a method by which the vehicle uses its sonar sensors to drive on a path parallel to a feature, such as a wall, railroad track or parked cars.

5.1. Feature selection and path determination

Figure 10 shows data collected in the local grid map when tracking cars parked on the right hand side of the road. As can be seen in the figure, the side of the cars facing the road is fairly well detected. Usually, sonar does not detect smooth surfaces very well because of specular reflections. However, in most real-world environments such as this one, there are no perfectly smooth surfaces. In this case, the sonar receives echo returns from corners and projections like door handles, mirrors, wheels, etc.

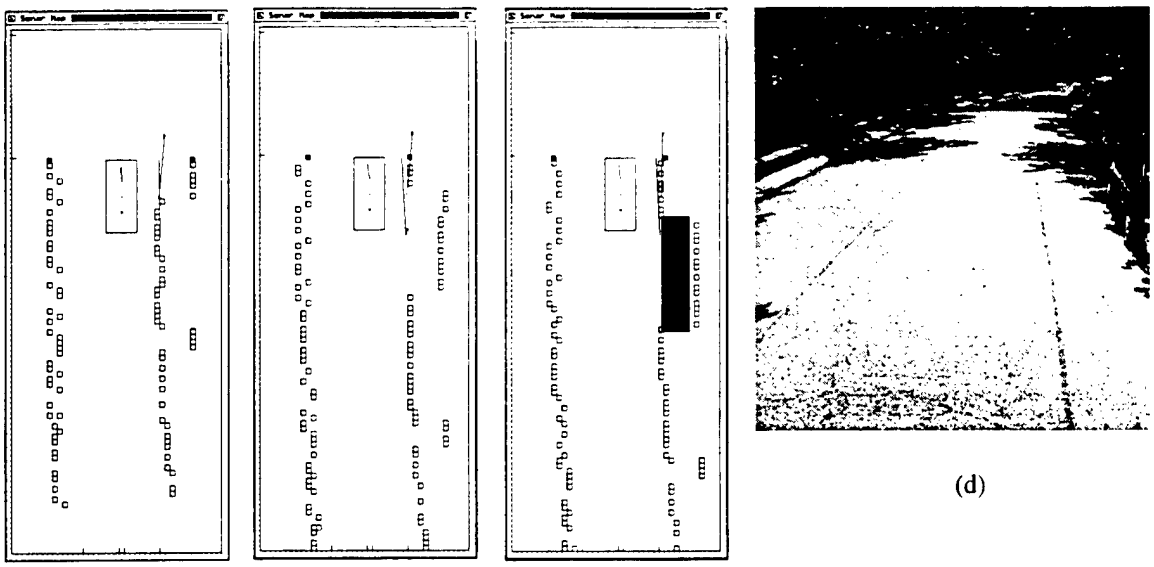


Figure 10: Searching for parking space: (a) Approaching gap, (b) Detecting gap, (c) Preparing vehicle for parking manoeuvre, (d) Typical street scene.

The vehicle is to drive on a path parallel to the curve formed by the parked cars, keeping a constant distance from the cars (usually around 2.5 m). Therefore the parameters of that curve have to be calculated first, using the data from the grid map. For reasons of computational simplicity and decreased noise sensitivity, a least-square fit of a straight line was chosen. All computations are performed with respect to the vehicle origin, which is at a fixed position in the grid map. Data points for the line fit are selected by choosing only data points that appear in a specific area in the map. Thus, for the environment represented in Figure 10, only the right half of the map is searched for data points. The **Position** parameter gives a data point in the map in terms of vehicle coordinates. Since the direction of vehicle motion is along the y-axis, a line parallel to the vehicle would have a slope of $m = \infty$ (Figure 3; Note that this coordinate system is defined by the vehicle's position system). To avoid this inconvenience, the vehicle coordinate system is rotated counterclockwise by 90°. All following computations are now performed with the transformed coordinates $y = -x_{old}$ and $x = y_{old}$. Therefore, the selected data points can now be represented as a discrete function $y_i = f(x_i)$. The sonar sensors sometimes return a spurious echo. These outliers generally degrade the performance of a least square fit. Hence, a median filter is applied first along x_i on the data points given by $y_i = f(x_i)$. The filter is applied twice, using window sizes three and five points. The two parameters of the straight line $y = mx + c$ can now be found by using standard formulae for a least-square fit for n data points (x_i, y_i) :

$$m = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2} \tag{8}$$

$$c = \frac{\sum y_i}{n} - m\frac{\sum x_i}{n} \tag{9}$$

Also, the standard error of estimate s_0 provides a measure of how well the data points could be fitted and is given by:

$$s_0 = \sqrt{\frac{\sum [y_i - (mx_i + c)]^2}{n-2}} \tag{10}$$

The data trajectory parameters m and c are stored and updated during each system cycle (Figure 7). The line can now be used by the controller to steer the vehicle (section 5.3.). To ensure a consistent steering response, the line fit also has to be checked for validity. It may happen that the sensors do not detect any features for some driving distance. One reason, in the case of parked cars for example, may be a gap between cars and no reflectors on the road edge. Here the line fit will produce no valid output. Therefore, if less than four data points are available, the output of the line fit is ignored. In this case the old path parameters are used and the vehicle will continue driving in the previously calculated direction until it encounters features again. Since the old path parameters are referenced to the vehicle position, they still have to be updated to compensate for vehicle movement during one system cycle. The position and orientation increment during one system cycle is known from equation (1). Hence m is transformed as follows:

$$\varphi = \text{atan}(m), \qquad m_{new} = \tan(\varphi - \delta\varphi) \tag{11}$$

In order to obtain c_{new}, point P (x_1, y_1) is selected on the path where it intersects the y-axis (for convenience) as shown in Figure 11. Using equations (1) and (2) the coordinates of P in the new coordinate system (x_{1new}, y_{1new}) can be calculated. Hence c_{new} can be calculated by

$$c_{new} = y_{1new} - m_{new}\, x_{1new} \tag{12}$$

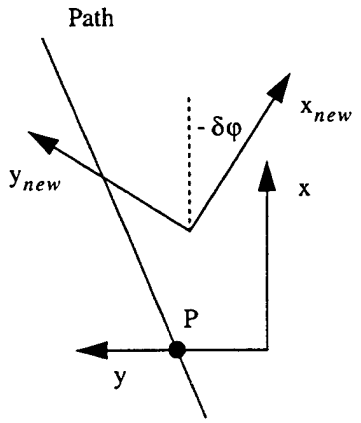


Figure 11: Path parameter transformation.

5.2. Path filtering

The method described above worked well in an environment that provided good reflectors and continuous smooth features. However, especially in the case of parked cars, problems arise when gaps are encountered or reflectors that do not belong to the feature being tracked are nearby. A typical situation is shown in Figure 12. In the least-square fit sequence 1 - 4, only line fits 1 and 4 track the feature. Type A data does not belong to the feature being tracked, type B data is due to corner effects at a gap and type C data is a noisy range measurement. Type B data can lead to an undesired least-square fit as shown by line fits 2 and 3. In order to reduce these errors, obtain a smoother steering response and make the system more robust against outliers, the values obtained from the least-square fit are filtered and path parameters are updated by merging new information with past values.

An initial noise reduction is achieved by selecting data points only from a small window in the local grid map. As we have some knowledge about the environment the vehicle is driving in, we can predict up to a certain degree where we should look for valid data in the map. This means in practice that only data points within a certain distance from the data trajectory are taken. This procedure removes outliers of type A as shown in Figure 12. Furthermore a point is selected only if it is within a certain distance and direction from previously selected adjacent points. This method removes outliers of type C and ensures that most points are already grouped close to a line segment.

Again a straight line is fitted using equation (9). For each distance δs_i traveled, we obtain a new value of gradient m_i. The change in gradient is then given by

$$\delta m_i = m_i - m_{i-1} \tag{13}$$

and $\delta\varphi_i$ is the corresponding change in vehicle orientation during that interval. Since not all type B data is removed, there still may remain a problem with sudden large changes in orientation as shown by the least square fit sequence 1 - 4 in Figure 12.

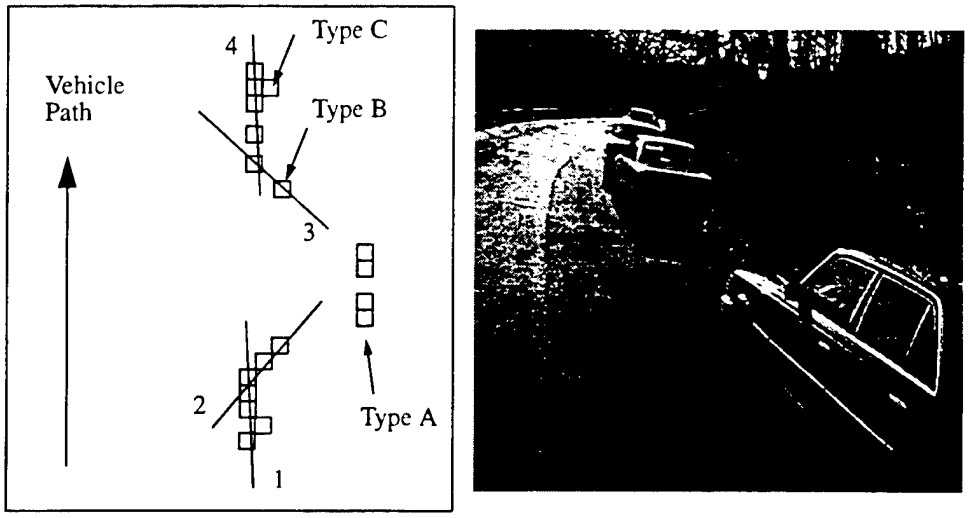


Figure 12: Removing outliers from least square fits:
(a) Map display, (b) Typical corresponding scene.

These sudden changes in orientation basically appear as median noise. They can be filtered out by putting current and past readings of δm into a buffer of N values, passing a median filter across and then averaging over N values. The buffer is implemented as an ordered set,

$$m_{Buffer} = \{\delta m_i,\ \delta m_{i-1},\ \delta m_{i-2},\ \ldots,\ \delta m_N\} \tag{14}$$

Initially, all buffer elements are set to zero, which means that it is assumed that the vehicle so far has driven exactly parallel to the data trajectory. During each update sequence, all elements are shifted right by one, the last element being discarded and the current result replacing the first element. Median filtering is achieved by replacing element δm_{i-k} with a median value, where $2k + 1$ is the maximum possible window size of the filter. Since values towards the right in the buffer represent increasingly earlier points in time/distance, successive applications of a median filter can be achieved by replacing elements δm_{i-k-t} with the filtered value, where t is the discrete shift in time/distance. The new value $m_{Path}^{\{new\}}$ of the path parameter is then computed, compensating also for change in vehicle orientation during the averaging interval,

$$m_{Path}^{\{new\}} = \frac{1}{N} \cdot \sum_{k=1}^{N} \tan\left(\operatorname{atan}(\delta m_k) - \sum_{i=1}^{k} \delta\varphi_i\right) \tag{15}$$

m_{Path} is then updated by merging the current and the new value for m_{Path} using a weighted average,

$$m_{Path}^{\{t+1\}} = \frac{m_{Path}^{\{new\}} + w \cdot m_{Path}^{\{t\}}}{1+w} \quad \text{, where } w \geq 0 \tag{16}$$

In a similar fashion as described above, the path parameter c_{Path} is updated. The weight w and the number of values N control how close the path parameters m_{Path} and c_{Path} should follow the actual data points. If a lot of noise is present in the data, the path parameters should be influenced only slightly, whereas if little noise is present, the data should be followed closely. An estimate of the noise is given by the standard error s_0 from equation (10) and w and N are adjusted accordingly; i.e. small w and N for a small standard error and large w and N for a large standard error. The actual values were determined empirically. As a result, the steering of the vehicle is now much smoother. From m_{Path} we can also obtain an estimate of the road curvature Υ over a certain distance Δs traveled, using

$$\Upsilon = \frac{\Delta\phi}{\Delta s} \quad \text{, where } \Delta s = \sum \delta s_i \text{ , } \quad \Delta\phi = \operatorname{atan}(m_{Path}^{\{t+1\}}) - \operatorname{atan}(m_{Path}^{\{t\}}) + \sum_{N} \delta\varphi_i \tag{17}$$

A drawback of the method is its slow reaction to a relatively sudden change in road direction. This effect is also due to the short range of the ultrasonic sensors and the fact that almost no echoes are received at large angles of incidence. In the case of features being tracked on the right side, the system is able to handle curves to the right since data points slowly move away from the vehicle and the vehicle can follow with a slight delay. On the other hand, a curve to the left poses a problem because by the time the vehicle recognizes that it should change direction, it has usually come already too close to the feature and has no space left to make a sharp left turn anymore.

For this reason a monitor is added that monitors a particular area for objects towards the right hand side of the vehicle as indicated in Figure 3. If objects are detected in this area, then the bitvector representing inhibited arcs is checked for each occupied grid cell as described in section 4. If the resultant bitvector does not permit an arc as commanded by the path tracker, the obstacle avoidance procedure overrides the tracker. In this case, the closest permitted left turning radius is selected. This ensures that any part of a feature being tracked to the right of the vehicle will be avoided. In general, the following condition has to be true for all occupied grid cells i in the monitored area, using curvature:

$$Curv_{tracker} < \operatorname{Min}_i \, (Curv_{Lb})_i \tag{18}$$

Altogether, the system performs well when tracking a wall or a feature that changes curvature smoothly. When tracking parked cars it does not perform as well in curves and fails when the road curvature becomes sharp. The reason here is that often parked cars are not very well aligned and even on straight road stretches parked at different angles to each other. The maximum range of the sensors is simply too short to detect these configurations well enough. Figure 10 shows the vehicle tracking parked cars, detecting a gap and preparing for reverse parking.

5.3. Path-Tracking Algorithm

The actual path-tracking procedure used by the system is based on the *pure pursuit* tracking described in [1]. The goal to be achieved is to drive the vehicle on a path parallel at a constant distance D_{ref} to the data trajectory given by m and c. Since the Navlab cannot move in any arbitrary direction from a given position, a point on the vehicle has to be selected that will be kept at distance D_{ref} from the data trajectory. As the vehicle uses Ackerman steering, the front part of the vehicle experiences the largest displacement when the steering angle is

changed. Therefore, we do not use the vehicle position which is at the centre of the rear axle, but the position of the front part of the vehicle to be kept at D_{ref}. Using data trajectory parameters m and c, the spatial displacement ε and angular displacement β for points P_o or P_o^* from the desired path can be calculated (Figure 13):

$$\beta = \text{atan}(m) \tag{19}$$

since vehicle orientation is always fixed in the map.

$$\varepsilon = D_{act} - D_{ref} \tag{20}$$

$$D_{act} = y_F \cdot \cos\beta \ , \quad y_F = m \cdot x_F + c \tag{21}$$

where x_F is the distance between vehicle position and the point selected at the front part of the vehicle as reference for path tracking.

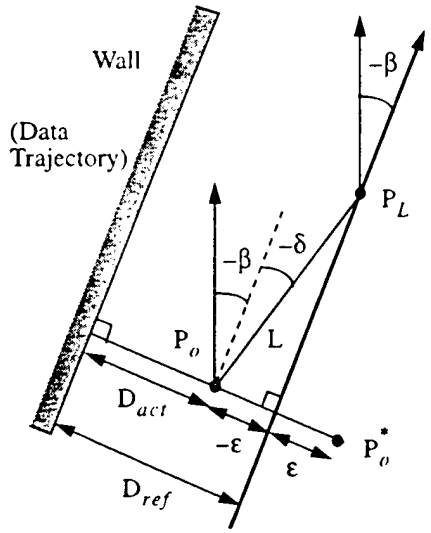


Figure 13: Path tracking parameters.

If the vehicle is now displaced by ε from the desired path, the *pure pursuit* method is used to drive the vehicle back onto the desired path. In this method, an initial goal point is selected on the desired path by the look ahead distance L (Figure 14). An arc is fitted between the current vehicle position and this goal point. The vehicle now moves a small distance during one system cycle and then a new goal point is selected. If the cycle time is small enough, a smooth path is executed that drives the vehicle back onto the desired path as shown in Figure 14. The path-tracking procedure works for data trajectories on either side of the vehicle. Only D_{ref} changes in sign.

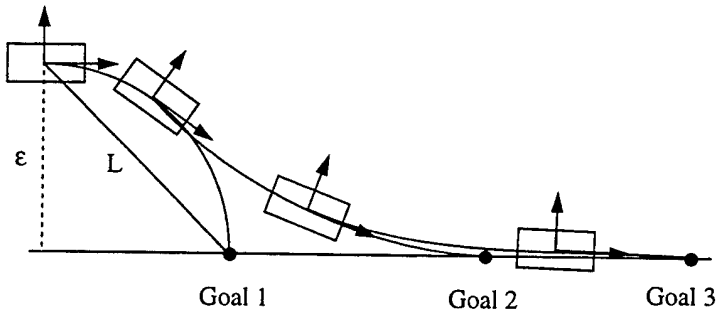


Figure 14: Pure pursuit.

The steering radius R during each cycle time can then be calculated by:

$$R = -\frac{L}{2\sin\phi}$$

$$\phi = \beta + \delta\ , \qquad \delta = \operatorname{asin}\frac{\varepsilon}{L} \tag{22}$$

If the vehicle drives on a narrow road with objects on both sides, keeping D_{ref} at a fixed value may not be appropriate as the vehicle may come too close to the other side. In this case we obtain D_{actR} and D_{actL} from the right and left side, respectively, and we can use this information to calculate a desired path that lies in the center between objects on both sides.

6. Parallel Parking

A system for parallel parking was implemented in order to demonstrate the performance of the sonar-based navigation system described above,. In that system, the vehicle can searches for a parking gap in a row of parallel parked cars and parks itself autonomously by using the procedures described in this section. This is good test problem because of the high accuracy required in the parking operation.

The vehicle starts out in the tracking mode, driving parallel at a constant distance to a row of parked cars (refer to section 5.1.) and searches for a parking gap as shown in Figure 10. If the sideways-looking sensors do not see any new objects that can be tracked, it is assumed that the vehicle has reached the edge of a potential gap. The procedure for parking the vehicle is then as follows (Figure 15).

Since the current sensors cannot see the curb or the end of the gap, the vehicle's driving direction is predicted from the orientations of the parked cars passed previously (Figure 15 (a), Position B - C). When the vehicle drives past the next car, it detects the end point of the gap. Both gap endpoints are tagged as shown in Figure 15, and the algorithm checks now whether there are any obstacles present within the gap and whether Gap length L_G is sufficient.

Since parking is done via a fixed path trajectory shape, the minimum length of the gap depends on several constant parameters and is given by equation (23). The vehicle is then aligned in parallel to the car in front of the gap (Position C - D).

$$L_G = 2R_{min}\sin\alpha + \frac{D_{ref} + 0.5W_G - 2R_{min}(1-\cos\alpha)}{\tan\alpha} - Y_P \tag{23}$$

where W_G is the width of gap (average width of a car), R_{min} is the minimum turning radius of the vehicle and

α is a given constant. Here $\alpha = 30°$. The values selected for Y_P and α ensure that the side of the vehicle will not hit the car parked in front of the gap.

If there is a lateral offset between the distance D_{ref} that the tracker kept from parked cars and the current lateral distance of the vehicle with respect to the car in front of the gap, then this offset is corrected by moving the vehicle an equal distance through two opposite equal arcs (Figure 15 (a), D - E). This means the vehicle changes its heading by an angle θ on the first arc and by an angle $-\theta$ on the second arc. Hence vehicle orientation is the same at position D and E. The angle θ depends on the lateral offset ε and can be calculated by,

$$\theta = \operatorname{acos}\left(1 - \frac{\varepsilon}{2R}\right) \tag{24}$$

The vehicle stops now and then drives straight in reverse until the vehicle position is within distance Y_P of the gap edge labeled 's' (E - F, see Figure 15 (b)).

For reverse parking, first the minimum turn radius R_{min} to the right is commanded until the vehicle heading has turned through an angle $-\alpha$. Then the vehicle drives straight again in reverse for a distance ΔS. Since α is a given constant for reasons explained previously, ΔS determines how close the vehicle is parked to the curb and is given by,

$$\Delta S = \frac{|D_{ref}| + 0.5W_G - 2R_{min}(1 - \cos\alpha)}{\sin\alpha} \tag{25}$$

Currently, the curb cannot be sensed and therefore the width of the gap W_G is assumed to be an average of the width of a typical car. Following, the minimum turn radius $-R_{min}$ to the left is commanded until the vehicle heading has turned through an angle α (Position F - G).

The vehicle now reverses its direction and in the last step, moves forward straight again until the gap between the vehicle and the car in front is approximately one meter (G - H). The vehicle has now parallel parked.

During the reverse driving part of the parking procedure, the sensors are not used and the vehicle path is computed from the data points previously collected in the map. In the last part of the parking procedure (G - H), the sensors are used again in order to detect the vehicle parked in front of the gap.

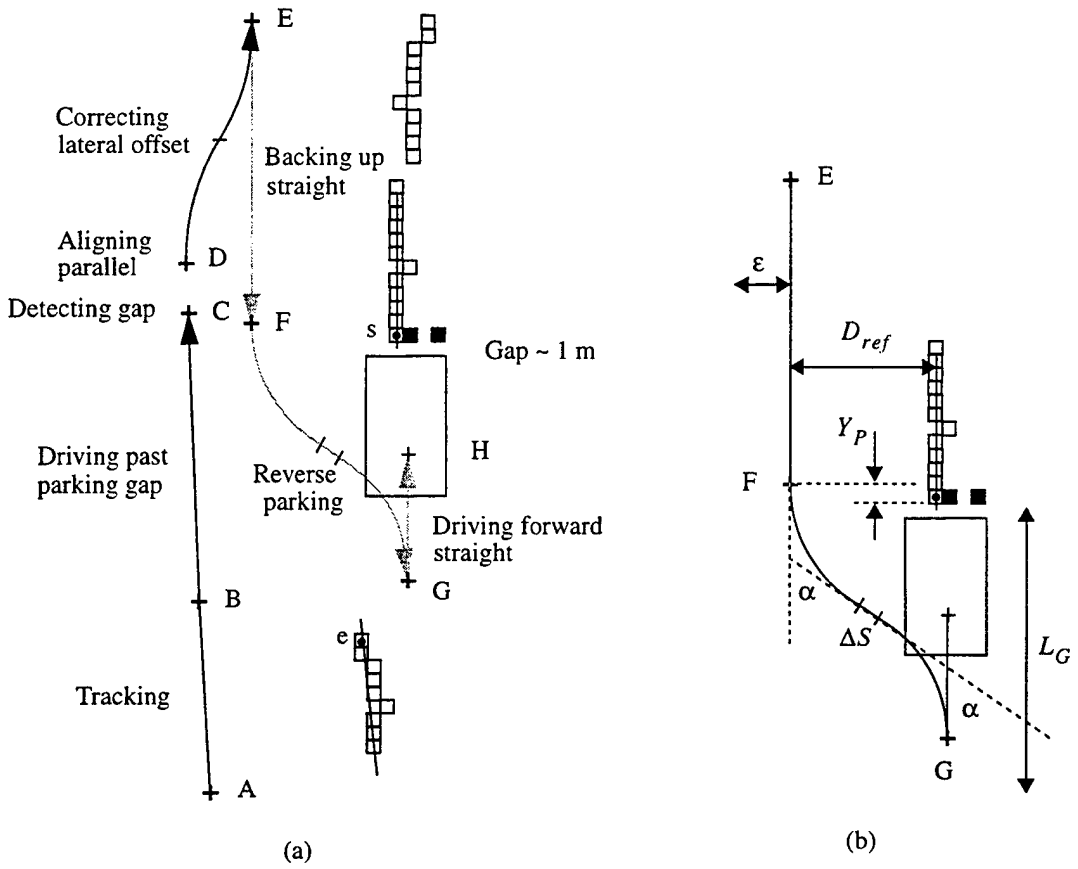


Figure 15: Parallel parking.

7. Results and Conclusions

Parking or docking maneuvers are important autonomous vehicle tasks. The system described has been successfully used to drive the Navlab parallel to parked cars in a city street, detect a parking space and autonomously park the vehicle.The sonar system also drove the Navlab on a dirt road next to a railroad track, using the railroad to guide the vehicle. The sonar is successfully integrated into two other systems that drive the Navlab. The first one is YARF ([5]), which drives the robot on city streets. Although YARF uses color vision for road following, it cannot detect if obstacles obstruct the vehicle's path and thus cannot adjust the velocity accordingly. The sonar system takes over that task and sends velocity commands to YARF via the TCX toolkit ([4]), using TCP/IP. The sonar is also integrated into the new architecture DAMN (Distributed Architecture for Mobile Robot Navigation) of the Navlab. The outputs of several modules that can drive the vehicle are used here to decide on an optimum path for the vehicle ([10]). The sonar module sends votes for a number of discrete steering arcs, voting against arcs that would result in a collision with an obstacle and voting for obstacle free arcs. The system then selects an obstacle free path to a goal point. Communication is again facilitated via the TCX toolkit.

The sonar system proved to work reliably in a variety of different situations. There were no major problems with false returns or sensor noise that could not be dealt with. One reason is that an outdoor environment like a road is generally less cluttered than most indoor environments where autonomous vehicles are used. Outdoor objects tend to be large, having usually enough corners and projections that reflect ultrasound well. Care has to be taken in avoiding reflections from the ground. This problem can be solved in most cases by mounting the

sensor high enough above the ground and pointing it slightly upward. The system can also be easily integrated with other vehicle navigation systems or adapted to other vehicles.

A drawback of using ultrasound in air is the limitation of range and data update due to high attenuation and low speed of sound. At low vehicle speeds this fact does not matter that much and the system works very well for slow speed maneuvers and parallel parking. However, the current system design is not limited to using only ultrasonic sensors.

A laser scanner (ERIM) was also successfully integrated into the system. Obstacles are detected from 3-D laser range images and stored in the grid map. Votes for discrete steering arcs are then computed and sent to the arbitrator in DAMN. This system has been used in cross-country navigation, driving the Navlab at speeds up to 3 m/s over a distance of 1 km ([6]).

Currently, we are planning to integrate a millimeter wave radar system with a range up to 200 meters for driving on highways and in city traffic.

Bibliography

[1] O. Amidi. *Integrated Mobile Robot Control.* Technical Report, Robotics Institute, Carnegie Mellon University. 1990.

[2] J. Borenstein and Y. Koren. Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments. In *Proc. IEEE Conference on Robotics and Automation.* pp 572-577. 1990.

[3] A. Elfes. A Sonar-Based Mapping and Navigation System. In *Proc. IEEE Conference on Robotics and Automation.* 1986.

[4] C. Fedor. *TCX, Task Communications User's Manual.* Internal Report, The Robotics Institute, Carnegie Mellon, 1993

[5] Karl Kluge and Charles Thorpe. Explicit Models for Robot Road Following. *Vision and Navigation: The Carnegie Mellon Navlab.* Kluwer Academic Publishers. Chapter 3. 1990.

[6] D. Langer, J. K. Rosenblatt and M. Hebert. A Reactive System For Off-Road Navigation. In *Proc. IEEE Conference on Robotics and Automation.* 1994.

[7] J. Leonard and H. Durrant-Whyte. Application of Multi-Target Tracking to Sonar-based Mobile Robot Navigation. In *Proc. IEEE Conference on Decision and Control.* 1990.

[8] V. Magori, H. Walker. Ultrasonic Presence Sensors with Wide Range and High Local Resolution. In *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control,* Vol. UFFC-34, No. 2. 1987.

[9] H. P. Moravec. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine,* v9#2. pp. 61-77. 1988.

[10] D. W. Payton, K. Rosenblatt and D. M. Keirsey. Plan Guided Reaction. *IEEE Journal on Systems, Man and Cybernetics.* 1990.

[11] C.E. Thorpe and J. Gowdy. Annotated Maps for Autonomous Land Vehicles. In *Proceedings of DARPA Image Understanding Workshop.* Pittsburgh PA. 1990.

[12] C.E. Thorpe, M. Hebert, Takeo Kanade and Steven Shafer. Toward Autonomous Driving: The CMU Navlab. Part I - Perception. In *IEEE Expert.* 1991.

STRIPE: Low-Bandwidth or High-Latency Vehicle Teleoperation System

1. Introduction

Driving a vehicle, either directly or remotely, is an inherently visual task. When heavy fog limits visibility, we reduce our car's speed to a slow crawl, even along very familiar roads. In teleoperation systems, an operator's view is limited to images provided by one or more cameras mounted on the remote vehicle. Traditional methods of vehicle teleoperation require that a real-time stream of images is transmitted from the vehicle camera to the operator control station, and the operator steers the vehicle accordingly. For this type of teleoperation, the transmission link between the vehicle and operator workstation must be very high bandwidth (because of the high volume of images required) and very low latency (because delayed images cause operators to steer incorrectly).

In many situations, such a high-bandwidth, low-latency communication link is unavailable or even technically impossible to provide. For example, current non-line-of-sight radio transmission technology does not provide the bandwidth required, and inter-planetary distances can delay communication by several minutes or more. A variety of image compression techniques have been developed to provide a higher frequency of images across low-bandwidth data links, but there is a limit to how much an image can be compressed before important data is lost.

As the frequency and immediacy of images transmitted to the operator decreases, the operator's ability to directly control the vehicle diminishes. At some point the operator's accuracy becomes so low that it is not practical to try to control a vehicle in such a manner.

When remote driving is required, but direct control is not possible, a "semi-autonomous" approach must be used. In semi-autonomous control, the operator provides the high-level decisions about where the vehicle should go, and the vehicle takes over the task of actually controlling the vehicle's steering. Because there is limited bandwidth and potential delays in the transmission system, the operator should be able to direct the vehicle based on a single image of the scene, and the vehicle should accurately follow the prescribed path for some distance while a new image is being transmitted. Other semi-autonomous teleoperation systems, which will be discussed in the next section, require stereo image data, or stray from the intended path on non-planar terrain. The STRIPE (Supervised TeleRobotics using Incremental Polyhedral Earth Reprojection) system is being developed to use only monocular image data and accurately follow a designated path on varying terrain (Figure 1).

In STRIPE, a single image from a camera mounted on the vehicle is transmitted to the operator workstation. The operator uses a mouse to pick a series of "waypoints" in the image that define a path that the vehicle should follow. These 2D waypoints are then transmitted back to the vehicle, where they are used to compute the appropriate steering commands while the next image is being transmitted. STRIPE does not require any advance knowledge about the geometry of the terrain to be traversed.

STRIPE is a unique combination of computer and human control. The computer must determine the 3D world path designated by the 2D image path and then accurately control the vehicle over rugged terrain. The human issues involve accurate path selection, and the prevention of disorientation, a common problem across all types of teleoperation systems. STRIPE has been successfully demonstrated on a variety of terrain types and we are beginning a series of experiments to improve the operator's ability to designate waypoints in the images accurately and to reduce operator disorientation.

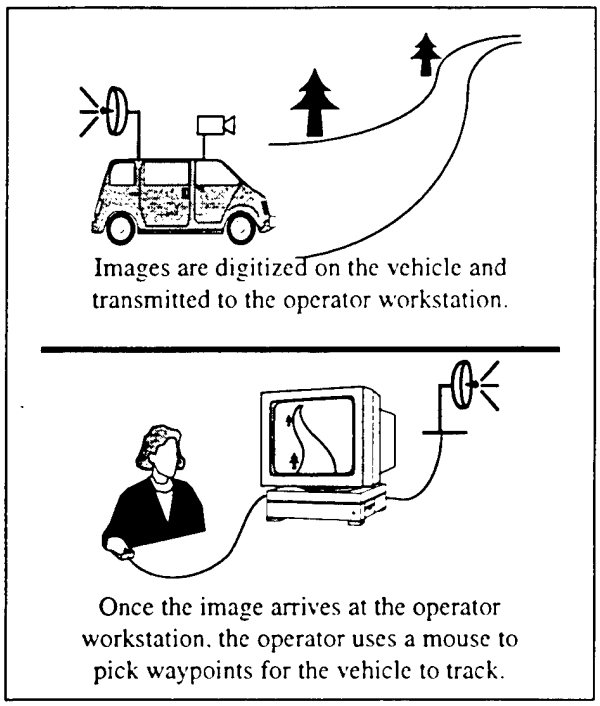


Figure 1: The basic STRIPE scenario.

2. Other Low-Bandwith High-Delay Systems

Depiero [2] demonstrated that human operators can directly and effectively control the steering of a remote vehicle if they are provided with approximately five images per second. The vehicle simulator community has found that delays of as little as a few hundred milliseconds in image data can be detrimental to an operators performance and physical well being [3]. Clearly, a transmission delay of one hour between image capture and display on the operator workstation is much too large to enable any sort of realistic steering of a teleoperated vehicle. Even a transmission delay of 1 minute per image is too long. It seems as though transmission rates of anything over several tenths of a second per image make direct driving impossible, and some kind of automatic vehicle control becomes necessary. This is not simply a theoretical problem. Consider a system for remotely operating a vehicle on Mars, for example, where transmissions to Earth can be delayed by about 15 minutes due to the distance between the planets. Or terrestrial systems that use current military standard tactical communication links, which only transmit 16 to 24 Kbits of data per second [2], a rate several orders of magnitude lower than that needed for real-time image transmission.

Computer-Aided Remote Driving (CARD) [6] is a system for semi-autonomous control of a remote vehicle. A stereo pair of images is transmitted to the operator from the vehicle. The operator uses a 3D mouse to pick waypoints on a stereo display. Since these waypoints are chosen in a 3D image, no 2D to 3D conversion is necessary, and no assumptions must be made about the shape of the terrain. These waypoints are transmitted back to the vehicle, where they are used to compute the appropriate steering directions. This method is very effective in situations where transmitting two images has approximately the same cost as transmitting a single image, as in high-bandwidth space applications. However, when the delay is not only due to distance, but also to low-bandwidth transmission, sending two images might take twice as long as sending a single image, and this may make the delays unacceptable.

The Feedback Limited Control System (FELICS) enables an operator to remotely drive a vehicle with a single image transmitted as infrequently as every 3.5 seconds [8]. A triangular "puck", a cursor that represents the

vehicle's location, is superimposed on the image, initially just in front of the vehicle's position. The operator uses a knob (to control heading) and a stick (to indicate speed) to "drive" the puck through the image, thereby establishing the path the vehicle should follow. At predefined intervals a marker is placed in the image along the puck's path to mark a "waypoint" (i.e. a point towards which the vehicle should steer).

In order to steer the vehicle, FELICS must convert the 2-D image waypoints into 3-D real-world points. This conversion is accomplished as follows: At the time of image capture, the location of the vehicle's "ground-plane", the plane on which the vehicle's wheels rest, is noted. Figure 2 shows a side view of the vehicle, the path to be traversed, and a vastly oversized pinhole camera representing the camera on the vehicle. Points on the image plane (i.e. 2-D waypoints) can be reprojected through the focus out into the world, and their intersection with the groundplane (represented by the x's) is considered to be their location in the 3-D world. The resulting 3-D waypoints are transmitted back to the vehicle, where they are used to control the steering.

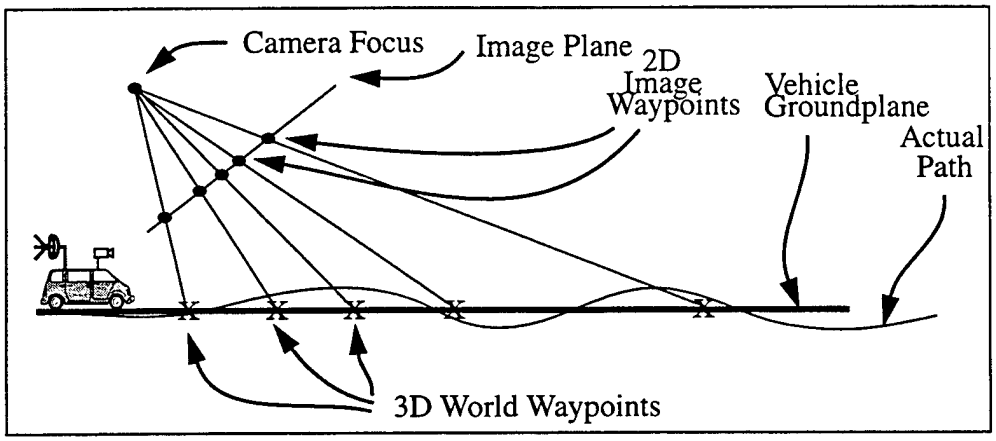


Figure 2: FELICS flat-earth reprojection.

While FELICS works well enough at 3.5 seconds per image to keep the vehicle on a wide road, its flat-earth projection method can cause errors on non-planar terrain. These errors can become increasingly problematic as image frequency decreases. For example, consider the scene depicted in Figure 3. The initial groundplane of the vehicle is horizontal, but the road ahead has a slight slope of 3 degrees. Suppose a sequence of points has been chosen by the operator, and a point 50 meters ahead of the vehicle on the road has been designated as the point where the vehicle should stop. With FELICS's simple flat-earth approach, all points are projected onto the current groundplane, and the vehicle will stop 25 meters ahead of its initial position, i.e. where the image point projects onto the current groundplane.

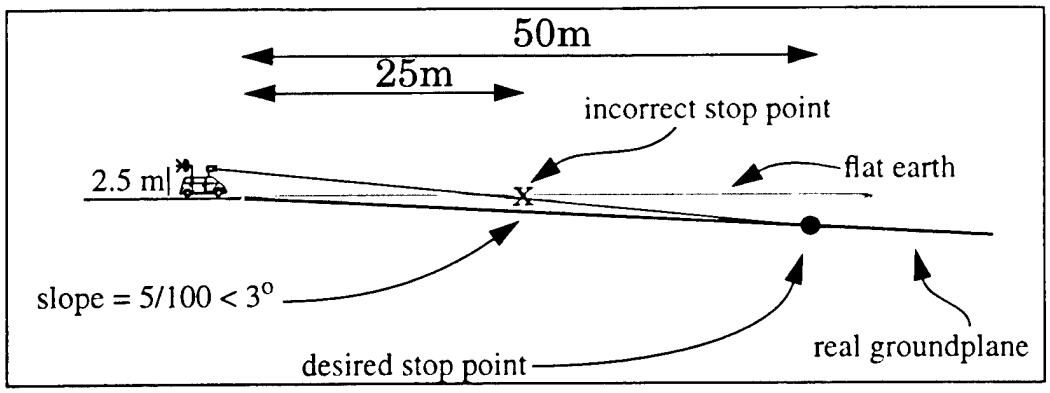


Figure 3: Flat earth reprojection.

3. The Current Stripe System

STRIPE [5] is a semi-autonomous teleoperation system that uses only a single image per cycle, but avoids the inaccuracies of FELICS's flat-earth reprojection.

The basic STRIPE system consists of three modules:

- The "Image Capture" module that runs on the vehicle.
- The "Operator Workstation" module that runs at the base station.
- The "Main STRIPE" module that runs on the vehicle.

A single image from a camera mounted on the vehicle is digitized by the Image Capture module. The image is then transmitted from the vehicle to the Operator Workstation module, where it is displayed on a monitor. The operator then uses a mouse to pick a series of "waypoints" in the image that designate the path the vehicle should follow (see Figure 1). These 2-D points are transmitted back to the Main STRIPE module on the vehicle.

In order to compute the vehicle's steering, the Main STRIPE module must convert these 2-D image waypoints into 3-D real-world points. Clearly, a simple flat-earth reprojection, such as that used by FELICS, is not sufficiently accurate.

When the 2-D waypoints are transmitted to the vehicle, they are initially projected onto the vehicle's current groundplane. The resulting 3-D waypoints are used to initiate steering of the vehicle, and the vehicle starts to move. Several times a second the vehicle re-estimates the location of its current groundplane by measuring vehicle position and orientation. The original image waypoints are then projected onto the new groundplane to produce new 3-D waypoints and the steering direction is adjusted appropriately. This reproject-and-drive procedure is repeated until the last waypoint is reached, or new waypoints are received.

For example, consider the side view of a road shown in Figure 4. The vastly oversized pinhole camera represents the camera on top of the vehicle, with the 2-D waypoints on the image plane. The intersection of the projections of each of the 2-D waypoints with the actual terrain is the location of the "actual" 3-D waypoint the operator intended the vehicle to track.

Initially, the STRIPE Main module reprojects the 2-D waypoints onto the vehicle groundplane, as shown in

Figure 5. After the vehicle has begun to move, it updates its prediction of the waypoints by reprojecting the original waypoints onto the new groundplane as shown in Figure 6. Figure 7 shows the vehicle after it has passed several of the waypoints.

STRIPE has no advance knowledge of the 3-D locations of all of the waypoints. However, as the vehicle approaches a particular waypoint, the vehicle's groundplane becomes an increasingly accurate approximation for the plane that the waypoint lies on. By the time the vehicle needs to steer based on that particular waypoint, it has a precise knowledge of where that point lies in the 3-D world. Note that all of this movement was based on a single image (and so our pinhole camera does not move throughout this segment).

It is important to emphasize that the flat-earth assumption that STRIPE makes is very different from standard flat-earth systems. Because the vehicle module is directly in charge of steering the vehicle, it does not need to plan all of the steering commands it will use in advance. Instead, it can compute the steering direction as it needs it. In the STRIPE path tracking algorithm, a "look-ahead distance" is a predefined constant value. The vehicle steers by finding the point on the desired path that is approximately one look-ahead distance away from the vehicle, and heading towards that point.

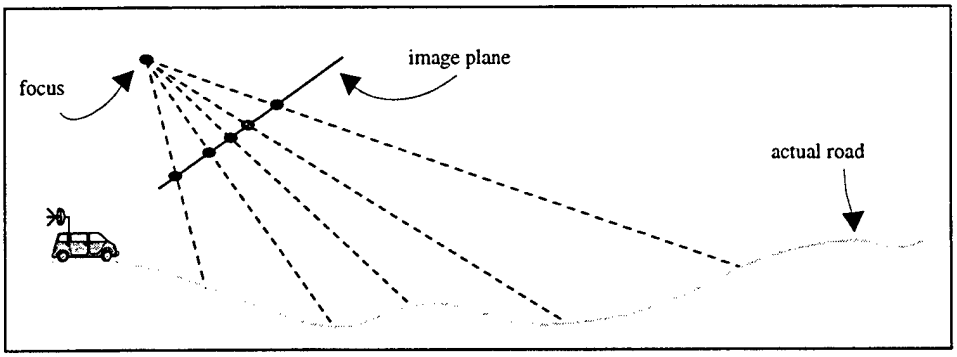


Figure 4: Side view of a road and the corresponding image, with designated waypoints. The giant pinhole camera represents the camera on top of the vehicle. Points on the image plane of the giant camera represent the 2-D waypoints chosen by the operator. The dotted lines show where those image points project onto the actual terrain. The intersection of each of the dotted lines with the actual road is the "correct" location of the 3-D waypoint.

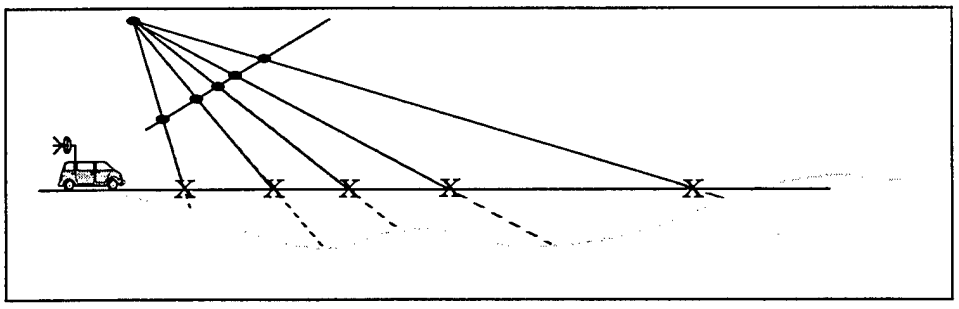


Figure 5: The 2-D waypoints are initially projected onto the vehicle's groundplane. The x's represent the location of the projected point.

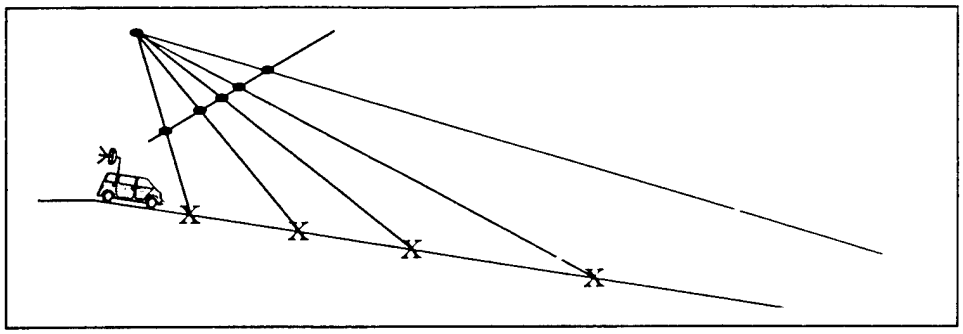


Figure 6: The same waypoints are reprojected onto the vehicle's new groundplane.

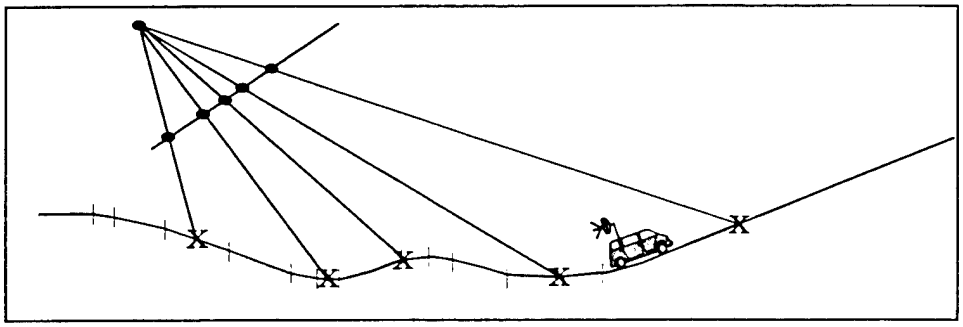


Figure 7: The vehicle continues its reproject-and-drive procedure. The path behind the vehicle has been divided into small planar regions.

How does STRIPE's local flat-earth assumption compare with the global one? Again, consider the scenario depicted in Figure 3. In the STRIPE scenario, we would stop at the appropriate point. Suppose we had a look-ahead distance of 5m. Initially, STRIPE would predict that a point that is really 5.56m ahead on the road was only 5m ahead, and begin to drive. However, as soon as the vehicle had rolled onto the new groundplane, subsequent predictions would be correct, and the vehicle would stop accurately at the stop point. This example is particularly illustrative of the problems that even a very mild change in groundplane orientation can cause. By restricting one's flat earth assumption to the look-ahead distance, the accuracy can be significantly improved. The "Main STRIPE" module on the vehicle runs at about 80 Hz on a Sun Sparc 10.

4. Investigating the Interface

4.1. Operator Difficulties

Experience with the more traditional high-bandwidth and low-delay teleoperation shows that operators have a tendency to be disoriented and become lost easily, even with landmarks and a map of the area [1][7]. In our case, even with low-bandwidth and high-delay teleoperation, the problem still exists, and the low frequency of images may actually aggravate it. Because there is a significant delay between images, operators cannot use the image flow to get a feel for the camera orientation. Without knowing the orientation of the camera relative to the vehicle, there is no way to determine where any given point in the image is relative to the vehicle.

For example, consider an on-road driving scenario. With the camera pointing straight ahead, the operator decides to make a left turn at an upcoming junction. When the vehicle is still some distance from the junction, the images sent back from the camera pointing straight ahead provide a view of the current road as well as the junction, and the operator easily picks points to move the vehicle towards the junction. When the vehicle gets closer to the junction, the camera must be panned to the left to see a better view of the road behind the junction, and an image is digitized. That image appears on the operator work station along with a number on the control panel indicating the camera pan and tilt. New operators seem to have difficulty using the numerical pan/tilt values to help interpret the image that they see. A typical operator comment is, "If I pick this point in the image, how sharply will the vehicle turn?" The problem seems to be one of understanding which point in the real world corresponds to a chosen point in the image. This does not seem to be as much of a problem in higher-bandwidth teleoperation systems, probably because the more frequent display of new image data provides the operator with a better sense of the vehicle's motion relative to the camera direction.

Figure 8: An "easy" road.

Figure 9: A "difficult" road.

Fixing the orientation of the camera on the vehicle does not solve the problem of choosing an appropriate path. For example, consider the images in Figure 8 and Figure 9. Both are images taken from a vehicle driving along a road with a fixed camera. In Figure 7, the path to choose to continue driving along the road is quite clear. However, Figure 9 was taken at an intersection where a single image provides few clues as to where to steer.

4.2. Improving the Interface

There is almost no empirical data available on the performance of low-bandwidth high-delay systems. In order to improve the performance of novice users, as well as create a standard of comparison for other systems, a series of empirical tests was developed to help determine how different variables affect an operator's skill at the teleoperation task.

The variables that will be studied include bandwidth, latency, field of view, resolution vs. image frequency, and color images vs. monochrome. Some of these have been studied in the context of high-bandwidth and low-delay teleoperation systems [4][7], but there is no way to extrapolate those results to the low-bandwidth and high-delay scenario.

Bandwidth and Latency - At the core of the system are the difficulties encountered by a low-bandwidth and high-delay transmission link. The maximum bandwidth is limited by that of the cellular modems that we are using for operator to vehicle communication (9600 bps), and we will also perform tests at reduced bandwidths. Latency (over and above delays due to restricted bandwidth) will be varied from 0s to 120s.

Vehicle Test Course - It is important to conduct these tests over a variety of test courses, both to determine the overall robustness of the system, as well as the influence of the course on other variables. We have developed a set of three test courses that are designed to test an operator's skill at obstacle avoidance, map following, and following "obvious" paths. The type of test course has been shown to have an influence in some tests of more traditional teleoperation systems. For example, Glumm and colleagues [4] found that operators preferred using a lens with a wider field of view on their obstacle avoidance test course.

Field of View - It seems likely that the influence of the field of view of the camera on a STRIPE operator will be very different from that of an operator of a traditional vehicle teleoperation system. In traditional systems, as in simulators, operators report feelings of discomfort and disorientation [3][4]. In a low-bandwidth and high-delay system, the fact that the images are discrete appears to eliminate these problems.

Operator Interface - The current STRIPE operator interface consists of three windows. One window shows the image transmitted from the vehicle, the second is a message window that provides the operator with instructions, and the third is a control panel that allows the operator to adjust camera angle, and start and stop the system (Figure 10).

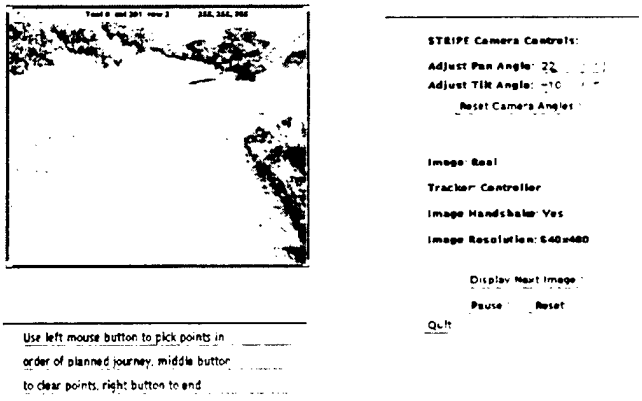


Figure 10: The current STRIPE operator interface.

To attempt to reduce the problems of disorientation, and to improve an operator's ability to understand the orientation of the camera with respect to the vehicle, we have developed two additional interfaces.

The idea behind the dashboard interface (Figure 11) is that individuals might better understand the orientation

of the camera with respect to the vehicle if they see the image as if through the window of the vehicle. Camera angle could be adjusted by using the mouse to move the outline of the image from left to right or by entering a numerical value on the control panel.

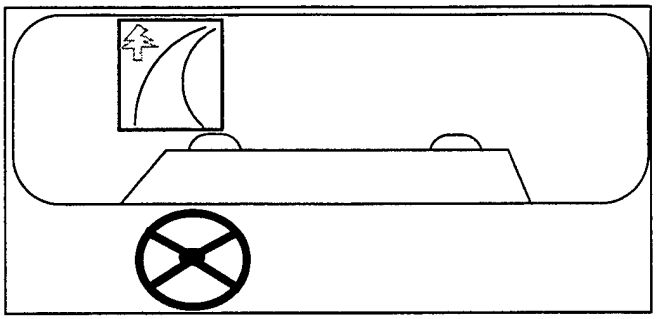


Figure 11: Dashboard interface.

The compass interface (Figure 12) presents an overhead view of the vehicle with the camera on top, and lines indicating the field of view of the camera. The orientation of the camera can be altered by using the mouse to grab one of the field of view lines and move them (the lines move together) or by entering a number on the control panel. The orientation of the vehicle would remain fixed, regardless of the vehicle's actual orientation.

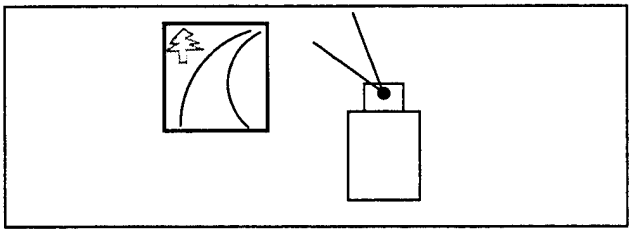


Figure 12: Compass interface.

Vehicle Control System - In addition to studying the standard STRIPE control system, we will study an operator's ability to use a steering wheel to directly control the vehicle across low-bandwidth and high-latency links. We will also test how quickly and accurately a normal driver in the vehicle can perform the given tasks.

Resolution vs. Image Frequency - When considering low-bandwidth data transmission, there is a trade-off between image resolution and frequency. Lower-resolution images can be transmitted with higher frequency, but do not provide as much information for the operator. Higher resolution images provide more detail, but at the cost of reduced image frequency. We will test three different image resolutions: 640x480 pixels, 320x240 pixels, and 160x120 pixels, and their corresponding relative frequencies.

Color vs. Monochrome Image Data - Anecdotal evidence suggests that color can be useful for humans teleoperating vehicles on Earth. However, color data are typically two to three times the size of monochrome data, and would thus reduce image frequency.

Method - Clearly, attempting to implement a detailed study of all of the independent variables described above would be a mammoth task. Initially, we will perform small-scale tests of the various independent variables to determine which seem to have the most dramatic effect on an operator's ability to accurately control a remote vehicle. Following these tests, the two variables that seem to have the most dramatic effect on an individual's ability to accurately operate a remote vehicle will be combined with the various vehicle test courses, three alternate STRIPE interfaces, and bandwidth vs. latency, and a more detailed series of empirical tests will be performed.

5. Conclusion

STRIPE is a unique combination of computer control, artificial intelligence, and intelligent user interaction. An important factor in this approach is its technological simplicity and low cost. The technology required for high-bandwidth data transmission is often very complex and expensive. STRIPE uses only simple components: monocular video display, mouse input, inertial sensor, and minimal computing power. This research not only provides a working system for teleoperation across low bandwidth and high latency links, it will also provide important data about a human operators ability to accurately direct a vehicle under such conditions.

6. Bibliography

[1] Aviles, W.A., Hughes, T.W., Everett, H.R., Umeda, A.Y., Martin, S.W., Koyamatsu, A.H., Solorzano, M.R., Laird, R.T., and McArthur, S.P. Issues in Mobile Robotics: The Unmanned Ground Vehicle Program Teleoperated Vehicle (TOV). *Proceedings of Mobile Robots V*, SPIE. Boston, MA. 1990.

[2] DePiero, Frederick W., Noell, Timothy E., and Gee, Timothy F. A Video Transmission System for Low-Bandwidth Remote Driving. *Proceedings of the ANS Fifth Topical Meeting on Robotics and Remote Systems*. Knoxville, TN. 1993.

[3] Frank, Lawrence H., Casali, John G., and Wierwille, Walter W. Effects of Visual Display and Motion System Delays on Operator Performance and Uneasiness in a Driving Simulator. *Human Factors*. Vol. 30, No. 2. 1988.

[4] Glumm, Monica M., Kilduff, Patricia W., and Masley, Amy S. *A Study of the Effects of Lens Focal Length on Remote Driver Performance*. ARL-TR-25, Army Research Laboratory, Human Research and Engineering Directorate, Aberdeen Proving Ground, MD. 1992.

[5] Kay, Jennifer, and Thorpe, Charles. STRIPE: Supervised TeleRobotics Using Incremental Polygonal-Earth Geometry. *Proceedings of the 3rd International Conference on Intelligent Autonomous Systems*. Pittsburgh, PA. 1993.

[6] Lescoe, Paul, Lavery, David, and Bedard, Roger. Navigation of Military and Space Unmanned Ground Vehicles in Unstructured Terrains. *Proceedings of the Third Conference on Military Robotic Applications*. 1991.

[7] McGovern, Douglas E. *Experiences and Results in Teleoperation of Land Vehicles*. Sandia Report SAND90-0299, Sandia National Laboratories. Albuquerque, NM. 1990.

[8] Rahim, Wadi. Feedback Limited Control System on a Skid-Steer Vehicle. *Proceedings of the ANS Fifth Topical Meeting on Robotics and Remote Systems*. Knoxville, TN. 1993.

Chapter V: Path Planning

Introduction

This Chapter describes techniques for path planning, that is, for computing the optimal trajectory from the current position of the vehicle to the next goal position. In the simplest systems, a path is pre-computed and supplied to the system so that the vehicle only needs to track a recorded path represented, for example, by a set of closely spaced points. Computing such a path would require a complete map of the environment, which may not be available.

In a more realistic scenario, the vehicle is commanded to follow the arc that brings the vehicle closest to the line between the vehicle's current position and a goal point. A new command is generated at regular intervals until the vehicle reaches the goal. Using this technique, called pure pursuit, a complete recorded path is no longer necessary and the path of the vehicle is computed using the position of the final goal point, or the positions of a few intermediate goal points.

Because pure pursuit does not take into account the navigability of the path, it must be used with an obstacle-avoidance behavior, which modifies the path according to the obstacles discovered as the vehicle travels. We have demonstrated systems based on the combination of pure pursuit and obstacle detection, as described in Chapter VI.

There are two fundamental problems with this approach. First, there is no guarantee that the vehicle will ever reach the goal because the configuration of obstacles may cause the vehicle to enter a cul-de-sac, which prevents it from reaching the goal. Such a situation is unavoidable in any moderately complex environment. Second, even if the vehicle can reach the goal, its path is not optimal. Both problems are due to the fact that neither obstacle avoidance nor pure pursuit take into account the obstacle map recorded as the vehicle travels.

To address those two problems, a new path planning algorithm called D* was introduced. D* uses all the map information accumulated since the beginning of a mission to compute the optimal path from the current position of the vehicle to the final goal point. Based on a modification of the classical A* algorithm, it is able to re-evaluate the path several times a second, thus taking into account new sensory information as soon as it becomes available. Because it takes into account the complete obstacle map, D* is able to modify the vehicle's path in real time whenever an obstruction is encountered, for example the "cul-de-sac" example above.

The D* algorithm provides the fundamental planning capabilities for missions in complex environments in the absence of a prior map.Chapter VI describes an integrated system based on SMARTY and D* that demonstrated the effectiveness of this approach. The theoretical foundation of D* is described in the first part of this Chapter.

The first implementation of D* was intended for a single vehicle driving to a single goal point. In general, however, missions may involve several vehicles and multiple goal points. For that reason, we have extended D* to multiple goals and vehicles. The basis of the algorithms used in this case and the evaluation of the performance in simulation examples are described in the second part of this Chapter.

Recent extension of D* include the ability to define the bounds of the map dynamically. This allows the use of D* with arbitrarily large maps. A second addition is the focussed D* algorithm in which only a subset of the cells are evaluated for higher computational performance.

D*: Optimal and Efficient Path Planning for Partially-Known Environments

1. Introduction

The research literature has addressed extensively the motion planning problem for one or more robots moving through a field of obstacles to a goal. Most of this work assumes that the environment is completely known before the robot begins its traverse (see Latombe [4] for a good survey). The optimal algorithms in this literature search a state space (e.g., visibility graph, grid cells) using the distance transform [2] or heuristics [8] to find the lowest cost path from the robot's start state to the goal state. Cost can be defined to be distance travelled, energy expended, time exposed to danger, etc.

Unfortunately, the robot may have partial or no information about the environment before it begins its traverse, but it is equipped with a sensor that is capable of measuring the environment as it moves. One approach to path planning in this scenario is to generate a "global" path using the known information and then attempt to "locally" circumvent obstacles on the route detected by the sensors [1]. If the route is completely obstructed, a new global path is planned. Lumelsky [7] initially assumes the environment to be devoid of obstacles and moves the robot directly toward the goal. If an obstacle obstructs the path, the robot moves around the perimeter until the point on the obstacle nearest the goal is found. The robot then proceeds to move directly toward the goal again. Pirzadeh [9] adopts a strategy whereby the robot wanders about the environment until it discovers the goal. The robot repeatedly moves to the adjacent location with lowest cost and increments the cost of a location each time it visits it to penalize later traverses of the same space. Korf [3] uses initial map information to estimate the cost to the goal for each state and efficiently updates it with backtracking costs as the robot moves through the environment.

While these approaches are complete, they are also suboptimal in the sense that they do not generate the lowest cost path given the sensor information as it is acquired and assuming all known, a priori information is correct. It is possible to generate optimal behavior by computing an optimal path from the known map information, moving the robot along the path until either it reaches the goal or its sensors detect a discrepancy between the map and the environment, updating the map, and then replanning a new optimal path from the robot's current location to the goal. Although this brute-force, replanning approach is optimal, it can be grossly inefficient, particularly in expansive environments where the goal is far away and little map information exists. Zelinsky [15] increases efficiency by using a quad-tree [13] to represent free and obstacle space, thus reducing the number of states to search in the planning space. For natural terrain, however, the map can encode robot traversability at each location ranging over a continuum, thus rendering quad-trees inappropriate or suboptimal.

This paper presents a new algorithm for generating optimal paths for a robot operating with a sensor and a map of the environment. The map can be complete, empty, or contain partial information about the environment. For regions of the environment that are unknown, the map may contain approximate information, stochastic models for occupancy, or even a heuristic estimates. The algorithm is functionally equivalent to the brute-force, optimal replanner, but it is far more efficient.

The algorithm is formulated in terms of an optimal find-path problem within a directed graph, where the arcs are labeled with cost values that can range over a continuum. The robot's sensor is able to measure arc costs in the vicinity of the robot, and the known and estimated arc values comprise the map. Thus, the algorithm can be used for any planning representation, including visibility graphs [5] and grid cell structures. The paper describes the algorithm, illustrates its operation, presents informal proofs of its soundness, optimality, and completeness, and then concludes with an empirical comparison of the algorithm to the optimal replanner.

2. The D* Algorithm

The name of the algorithm, D*, was chosen because it resembles A* [8], except that it is *dynamic* in the sense that arc cost parameters can change during the problem-solving process. Provided that robot motion is properly coupled to the algorithm, D* generates optimal trajectories. This section begins with the definitions and notation used in the algorithm, presents the D* algorithm, and closes with an illustration of its operation.

2.1. Definitions

The objective of a path planner is to move the robot from some location in the world to a goal location, such that it avoids all obstacles and minimizes a positive cost metric (e.g., length of the traverse). The problem space can be formulated as a set of *states* denoting robot locations connected by *directional arcs*, each of which has an associated cost. The robot starts at a particular state and moves across arcs (incurring the cost of traversal) to other states until it reaches the *goal* state, denoted by G. Every state X except G has a *backpointer* to a next state Y denoted by $b(X) = Y$. D* uses backpointers to represent paths to the goal. The cost of traversing an arc from state Y to state X is a positive number given by the *arc cost* function $c(X, Y)$. If Y does not have an arc to X, then $c(X, Y)$ is undefined. Two states X and Y are *neighbors* in the space if $c(X, Y)$ or $c(Y, X)$ is defined.

Like A*, D* maintains an $OPEN$ list of states. The $OPEN$ list is used to propagate information about changes to the arc cost function and to calculate path costs to states in the space. Every state X has an associated *tag* $t(X)$, such that $t(X) = NEW$ if X has never been on the $OPEN$ list, $t(X) = OPEN$ if X is currently on the $OPEN$ list, and $t(X) = CLOSED$ if X is no longer on the $OPEN$ list. For each state X, D* maintains an estimate of the sum of the arc costs from X to G given by the *path cost* function $h(G, X)$. Given the proper conditions, this estimate is equivalent to the optimal (minimal) cost from state X to G, given by the implicit function $o(G, X)$. For each state X on the $OPEN$ list (i.e., $t(X) = OPEN$), the *key* function, $k(G, X)$, is defined to be equal to the minimum of $h(G, X)$ before modification and all values assumed by $h(G, X)$ since X was placed on the $OPEN$ list. The key function classifies a state X on the $OPEN$ list into one of two types: a $RAISE$ state if $k(G, X) < h(G, X)$, and a $LOWER$ state if $k(G, X) = h(G, X)$. D* uses $RAISE$ states on the $OPEN$ list to propagate information about path cost increases (e.g., due to an increased arc cost) and $LOWER$ states to propagate information about path cost reductions (e.g., due to a reduced arc cost or new path to the goal). The propagation takes place through the repeated removal of states from the $OPEN$ list. Each time a state is removed from the list, it is *expanded* to pass cost changes to its neighbors. These neighbors are in turn placed on the $OPEN$ list to continue the process.

States on the $OPEN$ list are sorted by their key function value. The parameter k_{min} is defined to be $min(k(X))$ for all X such that $t(X) = OPEN$. The parameter k_{min} represents an important threshold in D*: path costs less than or equal to k_{min} are optimal, and those greater than k_{min} may not be optimal. The parameter k_{old} is defined to be equal to k_{min} prior to most recent removal of a state from the $OPEN$ list. If no states have been removed, k_{old} is undefined.

An ordering of states denoted by $\{X_1, X_N\}$ is defined to be a *sequence* if $b(X_{i+1}) = X_i$ for all i such that $1 \le i < N$ and $X_i \neq X_j$ for all (i,j) such that $1 \le i < j \le N$. Thus, a sequence defines a path of backpointers from X_N to X_1. A sequence $\{X_1, X_N\}$ is defined to be *monotonic* if ($t(X_i) = CLOSED$ and $h(G, X_i) < h(G, X_{i+1})$) or ($t(X_i) = OPEN$ and $k(G, X_i) < h(G, X_{i+1})$) for all i such that $1 \le i < N$. D* constructs and maintains a monotonic sequence $\{G, X\}$, representing decreasing current or lower-bounded path costs, for each state X that is or was on the $OPEN$ list. Given a sequence of states $\{X_1, X_N\}$, state X_i is an *ancestor* of state X_j if $1 \le i < j \le N$ and a *descendant* of X_j if $1 \le j < i \le N$.

For all two-state functions involving the goal state, the following shorthand notation is used: $f(X) \equiv f(G, X)$. Likewise, for sequences the notation $\{X\} \equiv \{G, X\}$ is used. The notation $f(\cdot)$ is used to refer to a function independent of its domain.

2.2. Algorithm Description

The D* algorithm consists primarily of two functions: *PROCESS-STATE* and *MODIFY-COST*. *PROCESS-STATE* is used to compute optimal path costs to the goal, and *MODIFY-COST* is used to change the arc cost function $c(\cdot)$ and enter affected states on the *OPEN* list. Initially, $t(\cdot)$ is set to *NEW* for all states, $h(G)$ is set to zero, and G is placed on the *OPEN* list. The first function, *PROCESS-STATE*, is repeatedly called until the robot's state, X, is removed from the *OPEN* list (i.e., $t(X) = CLOSED$) or a value of -1 is returned, at which point either the sequence $\{X\}$ has been computed or does not exist respectively. The robot then proceeds to follow the backpointers in the sequence $\{X\}$ until it either reaches the goal or discovers an error in the arc cost function $c(\cdot)$ (e.g., due to a detected obstacle). The second function, *MODIFY-COST*, is immediately called to correct $c(\cdot)$ and place affected states on the *OPEN* list. Let Y be the robot's state at which it discovers an error in $c(\cdot)$. By calling *PROCESS-STATE* until it returns $k_{min} \geq h(Y)$, the cost changes are propagated to state Y such that $h(Y) = c(Y)$. At this point, a possibly new sequence $\{Y\}$ has been constructed, and the robot continues to follow the backpointers in the sequence toward the goal.

The algorithms for *PROCESS-STATE* and *MODIFY-COST* are presented below. The embedded routines are *MIN-STATE*, which returns the state on the *OPEN* list with minimum $k(\cdot)$ value (*NULL* if the list is empty); *GET-KMIN*, which returns k_{min} for the *OPEN* list (-1 if the list is empty); *DELETE(X)*, which deletes state X from the *OPEN* list and sets $t(X) = CLOSED$; and *INSERT*(X, h_{new}), which computes $k(X) = h_{new}$ if $t(X) = NEW$, $k(X) = min(k(X), h_{new})$ if $t(X) = OPEN$, and $k(X) = min(h(X), h_{new})$ if $t(X) = CLOSED$, sets $h(X) = h_{new}$ and $t(X) = OPEN$, and places or re-positions state X on the *OPEN* list sorted by $k(\cdot)$.

In function *PROCESS-STATE* at lines L1 through L3, the state X with the lowest $k(\cdot)$ value is removed from the *OPEN* list. If X is a *LOWER* state (i.e., $k(X) = h(X)$), its path cost is optimal since $h(X)$ is equal to the old k_{min}. At lines L8 through L13, each neighbor Y of X is examined to see if its path cost can be lowered. Additionally, neighbor states that are *NEW* receive an initial path cost value, and cost changes are propagated to each neighbor Y that has a backpointer to X, regardless of whether the new cost is greater than or less than the old. Since these states are descendants of X, any change to the path cost of X affects their path costs as well. The backpointer of Y is redirected (if needed) so that the monotonic sequence $\{Y\}$ is constructed. All neighbors that receive a new path cost are placed on the *OPEN* list, so that they will propagate the cost changes to their neighbors.

If X is a *RAISE* state, its path cost may not be optimal. Before X propagates cost changes to its neighbors, its optimal neighbors are examined at lines L4 through L7 to see if $h(X)$ can be reduced. At lines L15 through L18, cost changes are propagated to *NEW* states and immediate descendants in the same way as for *LOWER* states. If X is able to lower the path cost of a state that is not an immediate descendant (lines L20 and L21), X is placed back on the *OPEN* list for future expansion. It is shown in the next section that this action is required to avoid creating a closed loop in the backpointers. If the path cost of X is able to be reduced by a suboptimal neighbor (lines L23 through L25), the neighbor is placed back on the *OPEN* list. Thus, the update is "postponed" until the neighbor has an optimal path cost.

Function: PROCESS-STATE ()

```
L1   X = MIN-STATE( )
L2   if X = NULL then return -1
L3    k_old = GET-KMIN( ) ; DELETE(X)
L4   if k_old < h(X) then
L5     for each neighbor Y of X:
L6       if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
L7         b(X) = Y; h(X) = h(Y) + c(Y, X)
L8   if k_old = h(X) then
L9     for each neighbor Y of X:
L10      if t(Y) = NEW or
```

```
L11         (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
L12         (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
L13         b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L14   else
L15     for each neighbor Y of X:
L16       if t(Y) = NEW or
L17         (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
L18         b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L19       else
L20         if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
L21           INSERT(X, h(X))
L22         else
L23           if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
L24             t(Y) = CLOSED and h(Y) > k_old then
L25             INSERT(Y, h(Y))
L26   return GET-KMIN( )
```

In function *MODIFY-COST*, the arc cost function is updated with the changed value. Since the path cost for state Y will change, X is placed on the *OPEN* list. When X is expanded via *PROCESS-STATE*, it computes a new $h(Y) = h(X) + c(X, Y)$ and places Y on the *OPEN* list. Additional state expansions propagate the cost to the descendants of Y.

Function: MODIFY-COST (X, Y, cval)

```
L1   c(X, Y) = cval
L2   if t(X) = CLOSED then INSERT(X, h(X))
L3   return GET-KMIN( )
```

2.3. Illustration of Operation

The role of *RAISE* and *LOWER* states is central to the operation of the algorithm. The *RAISE* states (i.e., $k(X) < h(X)$) propagate cost increases, and the *LOWER* states (i.e., $k(X) = h(X)$) propagate cost reductions. When the cost of traversing an arc is increased, an affected neighbor state is placed on the *OPEN* list, and the cost increase is propagated via *RAISE* states through all state sequences containing the arc. As the *RAISE* states come in contact with neighboring states of lower cost, these *LOWER* states are placed on the *OPEN* list, and they subsequently decrease the cost of previously raised states wherever possible. If the cost of traversing an arc is decreased, the reduction is propagated via *LOWER* states through all state sequences containing the arc, as well as neighboring states whose cost can also be lowered.

Figures 1 to 3 illustrate the operation of the algorithm for a "potential well" path planning problem. The planning space consists of a 50 x 50 grid of *cells*. Each cell represents a state and is connected to its eight neighbors via bidirectional arcs. The arc cost values are small for the *EMPTY* cells and prohibitively large for the *OBSTACLE* cells.[1] The robot is point-sized and is equipped with a contact sensor. Figure 1 shows the results of an optimal path calculation from the goal to all states in the planning space. The two grey obstacles are stored in the map, but the black obstacle is not. The arrows depict the backpointer function; thus, an optimal path to the goal for any state can be obtained by tracing the arrows from the state to the goal. Note that the arrows deflect around the grey, known obstacles but pass through the black, unknown obstacle.

1. The arc cost value of OBSTACLE must be chosen to be greater than the longest possible sequence of EMPTY cells so that a simple threshold can be used on the path cost to determine if the optimal path to the goal must pass through an obstacle.

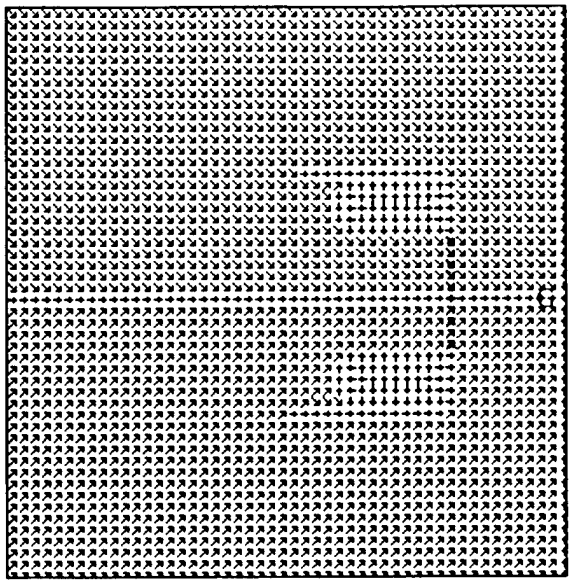


Figure 1: Backpointers based on initial propagation.

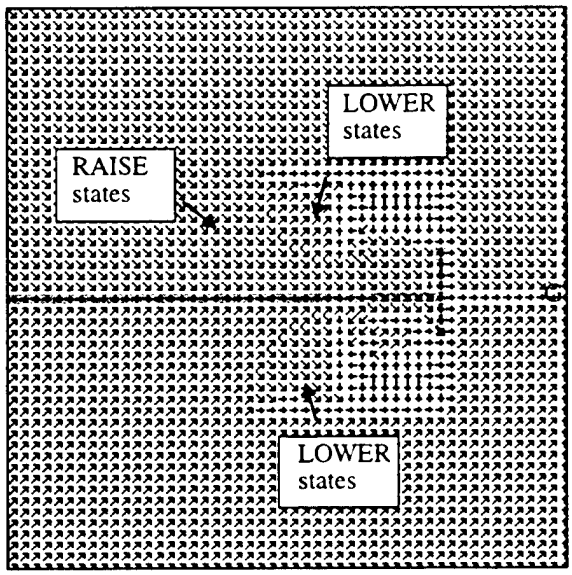


Figure 2: LOWER states sweep into well.

The robot starts at the center of the left wall and follows the backpointers toward the goal. When it reaches the unknown obstacle, it detects a discrepancy between the map and world, updates the map, colors the cell light grey, and enters the obstacle cell on the *OPEN* list. Backpointers are redirected to pull the robot up along the unknown obstacle and then back down. Figure 2 illustrates the information propagation after the robot has discovered the well is sealed. The robot's path is shown in black and the states on the *OPEN* list in grey. *RAISE* states move out of the well transmitting path cost increases. These states activate *LOWER* states around the "lip" of the well, which sweep around the upper and lower obstacles and redirect the backpointers out of the well.

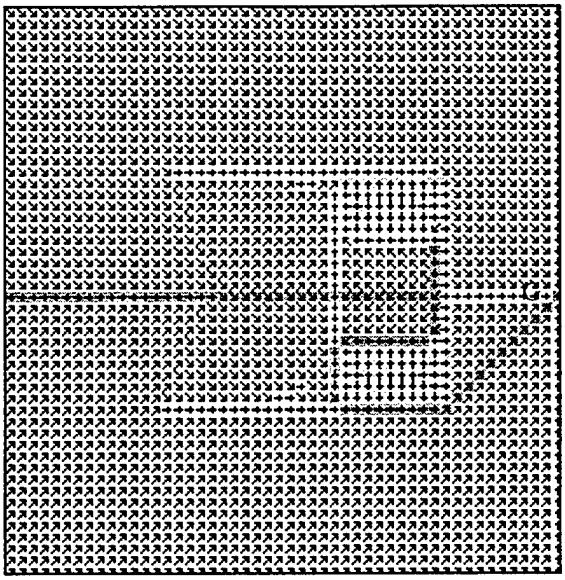


Figure 3: Final backpointer configuration.

This process is complete when the *LOWER* states reach the robot's cell, at which point the robot moves around the lower obstacle to the goal (Figure 3). Note that after the traverse, the backpointers are only partially updated. Backpointers within the well point outward, but those in the left half of the planning space still point into the well. All states have a path to the goal, but optimal paths are computed to a limited number of states. This effect illustrates the efficiency of D*. The backpointer updates needed to guarantee an optimal path for the robot are limited to the vicinity of the obstacle.

Figure 4 illustrates path planning in fractally generated terrain. The environment is 450 x 450 cells. Grey regions are fives times more difficult to traverse than white regions, and the black regions are untraversible. The black curve shows the robot's path from the lower left corner to the upper right given a complete, a priori map of the environment. This path is referred to as *omniscient optimal.* Figure 5 shows path planning in the same terrain with an optimistic map (all white). The robot is equipped with a circular field of view with a 20-cell radius. The map is updated with sensor information as the robot moves and the discrepancies are entered on the *OPEN* list for processing by D*. Due to the lack of a priori map information, the robot drives below the large obstruction in the center and wanders into a deadend before backtracking around the last obstacle to the goal. The resultant path is roughly twice the cost of omniscient optimal. This path is optimal, however, given the information the robot had when it acquired it.

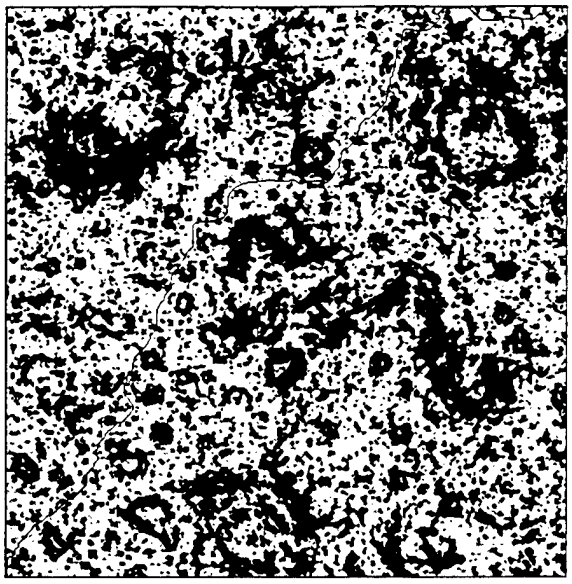


Figure 4: Path planning with a complete map.

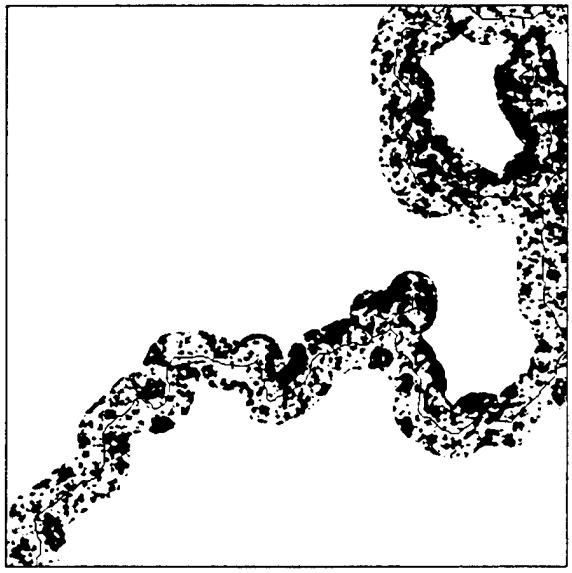


Figure 5: Path planning with an optimistic map.

Figure 6 illustrates the same problem using coarse map information, created by averaging the arc costs in each square region. This map information is accurate enough to steer the robot to the correct side of the central obstruction, and the resultant path is only 6 percent greater in cost than omniscient optimal.

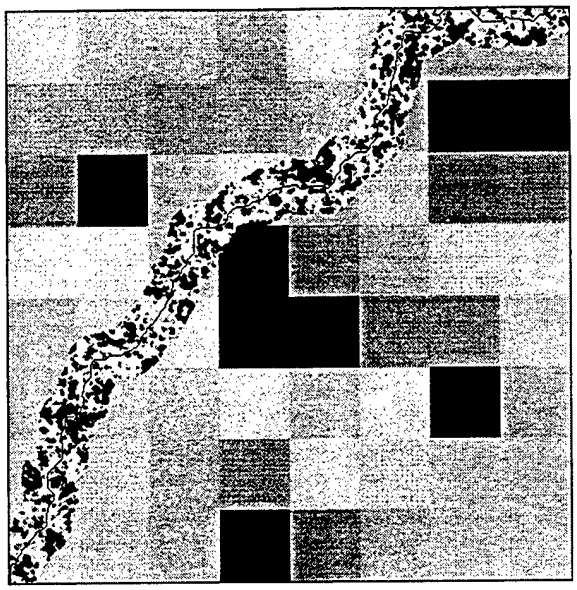


Figure 6: Path planning with a coarse-resolution map.

3. Soundness, Optimality, and Completeness

After all states X have been initialized to $t(X) = NEW$ and G has been entered onto the *OPEN* list, the function *PROCESS-STATE* is repeatedly invoked to construct state sequences. The function *MODIFY-COST* is invoked to make changes to $c(\cdot)$ and to seed these changes on the *OPEN* list. D* exhibits the following properties:

Property 1: If $t(X) \neq NEW$, then the sequence $\{X\}$ is constructed and is monotonic.

Property 2: When the value k_{min} returned by *PROCESS-STATE* equals or exceeds $h(X)$, then $h(X) = o(X)$.

Property 3: If a path from X to G exists, and the search space contains a finite number of states, $\{X\}$ will be constructed after a finite number of calls to *PROCESS-STATE*. If a path does not exist, *PROCESS-STATE* will return -1 with $t(X) = NEW$.

Property 1 is a soundness property: once a state has been visited, a finite sequence of backpointers to the goal has been constructed. Property 2 is an optimality property. It defines the conditions under which the chain of backpointers to the goal is optimal. Property 3 is a completeness property: if a path from X to G exists, it will be constructed. If no path exists, it will be reported in a finite amount of time. All three properties hold regardless of the pattern of access for functions *MODIFY-COST* and *PROCESS-STATE*.

For brevity, the proofs for the above three properties are informal. See Stentz [14] for the detailed, formal proofs. Consider Property 1 first. Whenever *PROCESS-STATE* visits a *NEW* state, it assigns $b(\cdot)$ to point to an existing state sequence and sets $h(\cdot)$ to preserve monotonicity. Monotonic sequences are subsequently manipulated by modifying the functions $t(\cdot)$, $h(\cdot)$, $k(\cdot)$, and $b(\cdot)$. When a state X is placed on the *OPEN* list (i.e., $t(X) = OPEN$), $k(X)$ is set to $h(X)$ preserve monotonicity for states with backpointers to X. Likewise, when a state X is removed from the list, the $h(\cdot)$ values of its neighbors are increased if needed to preserve monotonicity. The backpointer of a state X, $b(X)$, can only be reassigned to Y if $h(Y) < h(X)$ and if the sequence $\{Y\}$ contains no *RAISE* states. Since $\{Y\}$ contains no *RAISE* states, the $h(\cdot)$ value of every state in the sequence must be less than $h(Y)$. Thus, X cannot be an ancestor of Y, and a closed loop in the backpointers cannot be

created. Therefore, once a state X has been visited, the sequence $\{X\}$ has been constructed. Subsequent modifications ensure that a sequence $\{X\}$ still exists.

Consider Property 2. Each time a state is inserted on or removed from the *OPEN* list, D* modifies $h(\cdot)$ values so that $k(X) \leq h(Y) + c(Y, X)$ for each pair of states (X, Y) such that X is *OPEN* and Y is *CLOSED*. Thus, when X is chosen for expansion (i.e., $k_{min} = k(X)$), the *CLOSED* neighbors of X cannot reduce $h(X)$ below k_{min}, nor can the *OPEN* neighbors, since their $h(\cdot)$ values must be greater than k_{min}. States placed on the *OPEN* list during the expansion of X must have $k(\cdot)$ values greater than $k(X)$; thus, k_{min} increases or remains the same with each invocation of *PROCESS-STATE*. If states with $h(\cdot)$ values less than or equal to k_{old} are optimal, then states with $h(\cdot)$ values between (inclusively) k_{old} and k_{min} are optimal, since no states on the *OPEN* list can reduce their path costs. Thus, states with $h(\cdot)$ values less than or equal to k_{min} are optimal. By induction, *PROCESS-STATE* constructs optimal sequences to all reachable states. If the arc cost $c(X, Y)$ is modified, the function *MODIFY-COST* places X on the *OPEN* list, after which k_{min} is less than or equal to $h(X)$. Since no state Y with $h(Y) \leq h(X)$ can be affected by the modified arc cost, the property still holds.

Consider Property 3. Each time a state is expanded via *PROCESS-STATE*, it places its *NEW* neighbors on the *OPEN* list. Thus, if the sequence $\{X\}$ exists, it will be constructed unless a state in the sequence, Y, is never selected for expansion. But once a state has been placed on the *OPEN* list, its $k(\cdot)$ value cannot be increased. Thus, due to the monotonicity of k_{min}, the state Y will eventually be selected for expansion.

4. Experimental Results

D* was compared to the optimal replanner to verify its optimality and to determine its performance improvement. The optimal replanner initially plans a single path from the goal to the start state. The robot proceeds to follow the path until its sensor detects an error in the map. The robot updates the map, plans a new path from the goal to its current location, and repeats until the goal is reached. An optimistic heuristic function $\hat{g}(X)$ is used to focus the search, such that $\hat{g}(X)$ equals the "straight-line" cost of the path from X to the robot's location assuming all cells in the path are *EMPTY*. The replanner repeatedly expands states on the *OPEN* list with the minimum $\hat{g}(X) + h(X)$ value. Since $\hat{g}(X)$ is a lower bound on the actual cost from X to the robot for all X, the replanner is optimal [8].

The two algorithms were compared on planning problems of varying size. Each environment was square, consisting of a start state in the center of the left wall and a goal state in center of the right wall. Each environment consisted of a mix of map obstacles (i.e., available to robot before traverse) and unknown obstacles measurable by the robot's sensor. The sensor used was omnidirectional with a 10-cell radial field of view. Figure 7 shows an environment model with 100,000 states. The map obstacles are shown in grey and the unknown obstacles in black.

Figure 8 shows the results of the comparison for environments of size 1000 through 1,000,000 cells. The runtimes in CPU time for a Sun Microsystems SPARC-10 processor are listed along with the speed-up factor of D* over the optimal replanner. For both algorithms, the reported runtime is the total CPU time for all replanning needed to move the robot from the start state to the goal state, after the initial path has been planned. For each environment size, the two algorithms were compared on five randomly generated environments, and the runtimes were averaged. The speed-up factors for each environment size were computed by averaging the speed-up factors for the five trials.

The runtime for each algorithm is highly dependent on the complexity of the environment, including the number, size, and placement of the obstacles, and the ratio of map to unknown obstacles. The results indicate that as the environment increases in size, the performance of D* over the optimal replanner increases rapidly. The intuition for this result is that D* replans locally when it detects an unknown obstacle, but the optimal replanner generates a new global trajectory. As the environment increases in size, the local trajectories remain constant in complexity, but the global trajectories increase in complexity.

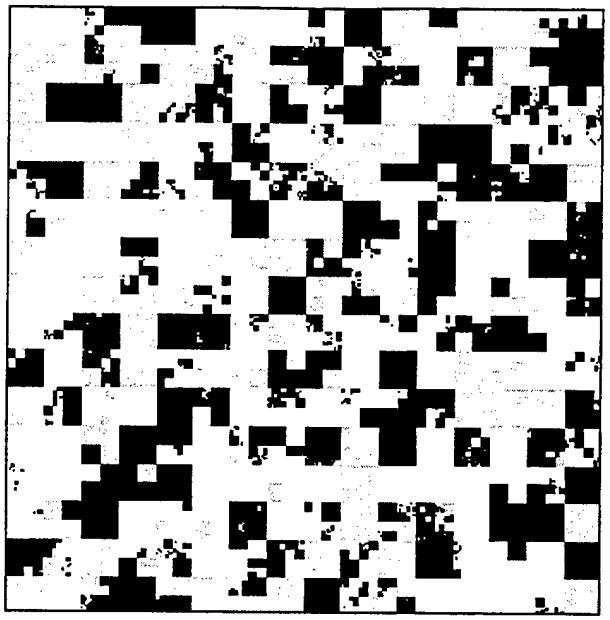


Figure 7: Typical environment for algorithm comparison.

Table 1: r

Algorithm	1,000	10,000	100,000	1,000,000
Replanner	427 msec	14.45 sec	10.86 min	50.82 min
D*	261 msec	1.69 sec	10.93 sec	16.83 sec
Speed-Up	1.67	10.14	56.30	229.30

Figure 8: Comparison of D* to optimal replanner.

5. Conclusions

5.1. Summary

This paper presents D*, a provably optimal and efficient path planning algorithm for sensor-equipped robots. The algorithm can handle the full spectrum of a priori map information, ranging from complete and accurate map information to the absence of map information. D* is a very general algorithm and can be applied to problems in artificial intelligence other than robot motion planning. In its most general form, D* can handle any path cost optimization problem where the cost parameters change during the search for the solution. D* is most efficient when these changes are detected near the current starting point in the search space, which is the case with a robot equipped with an on-board sensor.

See Stentz [14] for an extensive description of related applications for D*, including planning with robot shape, field of view considerations, dead-reckoning error, changing environments, occupancy maps, potential fields, natural terrain, multiple goals, and multiple robots.

5.2. Future Work

For unknown or partially-known terrains, recent research has addressed the exploration and map-building problems [6][9][10][11][15] in addition to the path-finding problem. Using a strategy of raising costs for previously visited states, D* can be extended to support exploration tasks.

Quad trees have limited use in environments with cost values ranging over a continuum, unless the environment includes large regions with constant traversability costs. Future work will incorporate the quad tree representation for these environments as well as those with binary cost values (e.g., *OBSTACLE* and *EMPTY*) in order to reduce memory requirements [15].

Bibliography

[1] Goto, Y., Stentz, A. Mobile Robot Navigation: The CMU System. *IEEE Expert*, Vol. 2, No. 4.1987.

[2] Jarvis, R. A.Collision-Free Trajectory Planning Using the Distance Transforms. *Mechanical Engineering Trans. of the Institution of Engineers*. Vol. ME10, No. 3. 1985.

[3] Korf, R. E. Real-Time Heuristic Search: First Results. *Proceedings Sixth National Conference on Artificial Intelligence*. 1987.

[4] Latombe, J.-C. *Robot Motion Planning*. Kluwer Academic Publishers. 1991.

[5] Lozano-Perez, T. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*. Vol. C-32, No. 2. 1983.

[6] Lumelsky, V. J., Mukhopadhyay, S., Sun, K. Dynamic Path Planning in Sensor-Based Terrain Acquisition. *IEEE Transactions on Robotics and Automation*. Vol. 6, No. 4. 1990.

[7] Lumelsky, V. J., Stepanov, A. A. Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment. *IEEE Transactions on Automatic Control*. Vol. AC-31, No. 11. 1986.

[8] Nilsson, N. J. *Principles of Artificial Intelligence*. Tioga Publishing Company. 1980.

[9] Pirzadeh, A., Snyder, W. A. Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1990.

[10] Rao, N. S. V. An Algorithmic Framework for Navigation in Unknown Terrains. *IEEE Computer*. 1989.

[11] Rao, N.S.V., Stoltzfus, N., Iyengar, S. S. A 'Retraction' Method for Learned Navigation in Unknown Terrains for a Circular Robot. *IEEE Transactions on Robotics and Automation*. Vol. 7, No. 5, October, 1991.

[12] Rosenblatt, J. K., Langer, D., Hebert, M. An Integrated System for Autonomous Off-Road Navigation. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1994.

[13] Samet, H. An Overview of Quadtrees, Octrees and Related Hierarchical Data Structures. In *NATO ASI Series*. Vol. F40, *Theoretical Foundations of Computer Graphics*. Springer-Verlag. 1988.

[14] Stentz, A. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*. Carnegie Mellon Robotics Institute Technical Report CMU-RI-TR-93-20. 1993.

[15] Zelinsky, A. A Mobile Robot Exploration Algorithm. *IEEE Transactions on Robotics and Automation*. Vol. 8, No. 6. 1992.

Dynamic Mission Planning for Multiple Mobile Robots

1. Introduction

Work in outdoor autonomous mobile robotics to date has been primarily concerned with maintaining robot safety, while moving in the environment. Robotic tasks, particularly in the outdoor realm, have been largely limited to path tracking with obstacle avoidance [7][8][9], or to single-robot / single-goal scenarios[19]. For mobile robotics to be effective in real-world applications, more than one robot must be able to share a potentially unknown workspaces and complicated missions with interdependencies between these robots must be feasible.

In many such applications, there exists a need to coordinate the action of robots to achieve a goal. Example applications include construction, military reconnaissance, warehouse automation, post-office automation, and planetary exploration. A mission in these applications could consist of a series of locations to be visited in any order by any robot, or perhaps a series of locations each robot should visit before returning to its recharging area. Existing systems for autonomous robots fail to permit the execution of complex missions for multiple robots because they do not adequately address the dynamic planning and mission planning aspects of these problems.

Dynamic planning is the process of efficiently updating the plans (i.e. changing each robot's plan) when further knowledge of the world is gained or when the control or local obstacle-avoidance system causes actual execution to differ from planned. Every perception system can introduce unavoidable noise into the system, "surprise" the planner with unexpected terrain, limit the motion of the robot due to a narrow field of view, and "change its mind" about terrain when viewed from a different position. These problems imply that estimated costs of mission plans may change frequently and that the robot may not follow a previously recommended path to the goal.

Mission planning is the process of determining what each robot should do to achieve the goals of the mission. Since environments encountered in the above domains are unstructured, the optimal mission plan may change dramatically due to new information received from any involved robot. Thus, a mission planner in an unstructured environment must operate in a dynamic fashion.

Mission planning problems have been addressed in the fields of AI planning and motion planning. Complete multi-robot systems have also been developed which coordinate particular multi-robot scenarios.

AI planners tend to be domain independent and frequently use heuristic search methods to determine an appropriate sequence of operators to reach the goal state. An excellent broad collection of papers on AI planning can be found in Readings in Planning [1]. A quintessential example of this type of symbolic system is STRIPS [5]. While such systems are very effective at finding a sequence of actions to reach a goal state and (as extended by Feldman and Sproull [13]) at minimizing the cost of accomplishing the task, there is a fundamental paradigm mismatch -- physically situated mobile missions are not easily linked with symbolic planners.

Motion planning systems have ranged from simple control schemes for local obstacle avoidance to omniscient schemes which reason about the entirety of the robot workspaces. Motion planners reason exclusively about movement, producing paths that avoid collision, maximize safety, and/or minimize distance traveled. Most traditional motion planning systems [10] (both for single and multiple [15] robot scenarios) are not immediately applicable to dynamic mission planning problems due to the constraints of a dynamic environment -- having to replan every time a map discrepancy is found or a robot fails is unacceptable. However, a dynamic planner capable of planning paths for a single robot and a single goal set has been designed and implemented by Stentz [17][18].

Several multi-robot systems that operate in a largely reactive fashion exist. Examples of these include

Mataric's work on flocking behaviors [13], Parker's ALLIANCE [14], and Georgia Tech's MissionLab [3]. While each of these systems is capable of moving robots safely through their environment, little attention is paid to optimizing robot motion, building world models that may be used by other robots, or solving missions that involve explicit choices between alternatives. Others, such as Rugg-Gunn [13] and Le Pape [12], use more traditional world models and a centralized planning framework to plan and execute complex missions. While these systems do optimize motion and permit complex plans, they are not designed for the efficient replanning necessary for operation in dynamic environments.

In summary, although existing AI Planners are effective at sequencing actions and Motion Planners are effective at keeping the robot safe while achieving a single goal; neither alone addresses optimizing the execution of tasks that may involve multiple goals and multiple robots. Current multi-robot systems either are not designed to reason about complex mission-oriented plans or are not effective in dynamic environments. This paper discusses a mission planning system designed to satisfy the demands of dynamic planning for multiple goals in unstructured environments as needed for a variety of mobile robot applications.

2. Problem Statement

The problem addressed can be stated as follows:

To support:

- motion planning and
- complex missions,

for:

- multiple robots,
- multiple concurrent goals, and
- dynamic environments,

using mobile robots which have:

- relatively open operational workspaces, and
- effective positioning, communication and perception,

permit:

- mission cost optimization,
- successful, optimized task completion, and
- dynamic replanning as world knowledge increases.

To address a general class of mission planning problems applicable to a variety of applications, we propose the following grammar in which missions may be expressed:

$$r \Rightarrow R \,\|\, (r_1 \wedge r_2) \,\|\, (r_1 \vee r_2)$$
$$g \Rightarrow G \,\|\, (g_1 \wedge g_2) \,\|\, (g_1 \vee g_2) \,\|\, (g_1 \rightarrow g_2)$$
$$m \Rightarrow M(r, g) \,\|\, (m_1 \wedge m_2) \,\|\, (m_1 \vee m_2) \,\|\, (m_1 \rightarrow m_2)$$

where R,G, and M stand for Robot, Goal and Move, respectively. Capital letters are constants, that is, a particular robot, goal, or mission. Lower case letters represent constants or other expressions. $A \wedge B, A \vee B, A \rightarrow B$ are interpreted as A "and", "or", "then" B, respectively.

To demonstrate how a robotic mission might be expressed in this grammar, consider the following expression:

$$M(R_1 \vee R_2, G_1) \rightarrow M(R_2, G_2 \rightarrow G_3)$$

This states that either Robot 1 or Robot 2 should go to Goal 1, then Robot 2 should visit Goal 2 and Goal 3 in that order. This could correspond to the data gathering scenario where either robot must visit one location (Goal 1), and then Robot 2 (which is perhaps equipped with a sensor that Robot 1 lacks) should go to Goals 2 and 3.

This paper studies a subset of this grammar to determine feasibility of a system that can address a large class of mission planning problems defined by the grammar above and constrained by the demands of dynamic planning. The problem consists of N robots initially located at different locations, M goals which must be reached (by any robot), and one base to which all robots must return. The performance metric is the longest distance travelled by any robot while visiting goals and then returning to base. In the grammar, this mission is expressed as:

$$M(R_1 \vee \ldots \vee R_N, G_1 \wedge \ldots \wedge G_M) \rightarrow M(R_1 \wedge \ldots \wedge R_N, G_{base})$$

The rest of this paper will discuss the technical approach to this prototype, its applicability to real tasks, the results obtained with this system, and directions for future work.

3. Technical Approach

Figure 1 shows the system data flow. There are three asynchronous processes operating. First, each Local Navigator (1 per robot) evaluates its current field of view and chooses a steering direction based on that field of view and its assigned goal. Second, the Dynamic Planners (D*'s) continually update the paths for all robots to all goals given the incoming map information from the Navigators. The path-cost information is provided to the Mission Planner, and the path information is provided to the Local Navigators. Finally, the Mission Planner continually updates the assignment of robots to goals based on the mission specification and informs each Local Navigator of its goal.

The benefit from the asynchronous operation of these processes is that each may cycle at a time scale appropriate to the assistance it provides towards optimizing[1] the mission performance. The Local Navigators cycle rapidly to ensure the robot never collides with anything in the environment. The Dynamic Planners cycle as quickly as possible, updating paths to the goals (used by the Local Navigators to bias choice of steering direction) and updating costs of the these paths (used by the Mission Planner to reassign robots and goals if a more efficient solution is found.) The Mission Planner can cycle slowest of all, as changes it makes typically have a large effect, and thus need not happen instantly. This is possible because the delay from when the relevant data arrives until the time when the change is computed is much smaller than the savings in mission completion time produced by the change. For example, if you are driving to work and you discover your typical route has heavy traffic, you can profitably take a minute or two to check a map before deciding to change your route if the change will save you significantly more than the 1-2 minutes you "wasted" driving in traffic while making the decision.

This system can be generalized easily because of the abstracted data passed between layers. Any local navigator which

- permits the description of the world as an undirected graph of locations (a 4/8-connected grid is

1. The notion of "optimal" in dynamic planning is different from that in static planning. In static planning, the environment is fully known and the optimal path, once planned, remains constant during execution. When the state of the world is not fully known, a mission plan is optimal if each robot R, at each point P_R along its traverse given the aggregate information available at P_R, follows the lowest cost path to its goal such that the mission is completed in a manner in which the cost of the longest path any robot must traverse is minimized, given the aggregate information available.

acceptable)

- possesses a method for translating perception data into state-to-state transition cost information, and
- accepts advice concerning a path to a goal,

can be used with this system This permits easy implementation of the planning system on a variety of navigators and their corresponding perception systems.

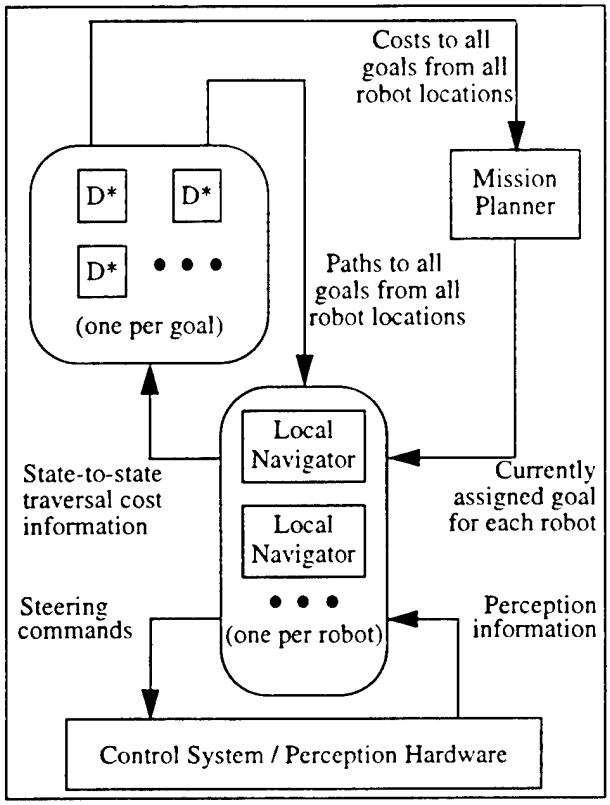


Figure 1: System Data Flow

4. Prototype System

4.1. Example Problem

Given the above system design, the only limitation in the complexity of feasible missions is the ability of the Mission Planner to understand them and find optimal solutions for them in a timely fashion. The example problem used in this prototype is a variety of the Multiple Travelling Salesman Problem (MTSP). The scenario which the Mission Planner optimizes is one where there are N robots each starting at different locations, M goals which must be visited by some robot, and 1 Base to which all robots must eventually return. The Mission is optimized with respect to getting all robots back to the Base as quickly as possible. All robots are homogenous in all respects.

This problem exhibits the most challenging aspect of designing a general Mission Planner - the problem it must optimize is NP-hard [11]. Virtually any mission involving multiple robots is going to require that the planner evaluate thousands (if not millions) of alternatives as quickly as possible. By picking a scenario with this order of difficulty, the feasibility and benefit of a such a solution can be tested. Any optimal solution to this problem is going to run in time exponential to the number of robots and/or goals. Thus, by picking a problem where the cycle time of the Mission Planner is much greater than the cycle time of the Local Navigator, it is possible to determine if a benefit from such a process can be gained, and thus, if a more general Mission Planner can be successfully implemented which optimizes more complex missions.

4.2. Prototype Operation

A number of simplifications were made to the design of the prototype system to permit it's rapid implementation without loss of applicability.

First, work by Stentz and Hebert [19] has demonstrated the feasibility of using D* coupled with a local navigator to drive a single real robot to a single goal. Thus, it was possible to move to a simulated environment that mimics the real environment without loss of applicability, as the identical software could be used to control the robot. This permitted testing multiple robot scenarios without having to build the communications infrastructure necessary to support a live test. The only changes from an existing system [19] which are necessary to make a live run would be to permit switching between which Dynamic Planner (D*) advises the Local Navigator based upon the decisions of the Mission Planner.

Second, the perception processing that converted range images into obstacle (cost) maps was eliminated by using a world which contained a binary obstacle map and a perception system which accessed this obstacle map directly.

Also, the processing was done on a single machine as a single process. The difficulties of using asynchronous processes had been solved for much of this application [19], therefore, it was possible to perform useful tests on an improved system without the additional complexity introduced by asynchronisity.

The prototype system functioned as follows. A world of random obstacles was generated, with M goals and N robots located randomly in this world. M dynamic planners are created, each tasked with maintaining the optimal path for N robots to the planner's assigned goal. On each cycle of the main loop of the prototype system, each dynamic planner performed a fixed amount of computation to update the current $N \times M$ costs of all robots to all goals based on the new perception information provided. Periodically, the Mission Planner exhaustively searched the alternative solutions to the mission using the latest computed costs. Based on the current best solution to the MTSP problem (which may have been computed on a earlier cycle if the Mission Planner was not active during the current cycle), each robot was then moved a single step in the direction of its goal based on the Dynamic Planner assigned to this goal. In this way, the planners cycled more slowly than the local navigator.

D* (Dynamic A*) is used as the dynamic planner to maintain the costs from each robot to each goal. The D* algorithm was developed to replan paths in real-time. D* produces an initial plan based on known and assumed information, and then incrementally repairs the plan as new information is discovered about the world. The repaired plan is guaranteed to be optimal and is functionally equivalent to replanning from scratch. The robot moves in such a way that at every point P along its traverse, the robot is following an optimal path from P to the goal, given all information known in aggregate at P. For large applications (i.e., environments with a million states or more), D* is more than 200 times faster than brute-force replanning. D* is fast because for many motion planning problems because new information is generally discovered by sensors carried on-board the robot and thus impacts the portion of the plan "local" to the robot's current state; therefore, most of the time only a small portion of the existing plan is affected.

The Mission Planner uses a straightforward exhaustive search of all possible alternatives for solving the MTSP

which comprises the mission statement. Segments of the mission can be limited to the paths planned independently by D* because the triangle inequality holds: D* always maintains the shortest path between two points, and there is always a path AC which costs no more than sum of the costs of AB and BC for any point B. Since most reasonable workspaces are relatively open, explicit methods for avoiding robot-robot collisions are not used. It is assumed local methods will be effective at avoiding such conflicts. As goals are reached they are removed from the Mission Planners search, thus speeding response time as the mission completes. Whenever the mission plan is changed (a robot needs to drive towards a different goal), a different D* instantiation is provides the driving direction for that robot. The only change in functionality necessary to make this prototype system able to solve more complex plans is an improved mission planning component.

5. Results

A straightforward scenario begins in Figure 2.

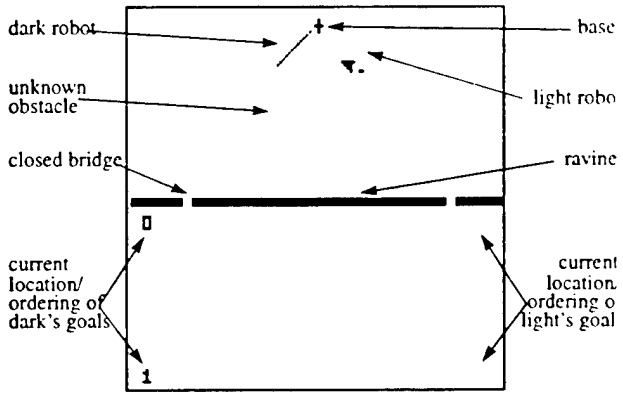


Figure 2: Simulation Step 1

In this example, N=2 and M=4. The cross in black at the top center is the base, and the left and right lines indicate the path of the "dark" and "light" robots. The Mission Planner has already planned that the dark robot should go to the goals on the left (numbered 0 and 1 in dark), and light should go to the goals on the right (numbered 0 and 1 in light). The black portion of the image corresponds to a ravine with two bridges that divide the environment in half. The grey regions, including in particular the blocked bridge on the left, are unknown obstacles in the environment. Thus, the dark robot believes it can go over the left bridge even though it is actually blocked. As the robots move and observe these unknown obstacles with their sensors, these areas are colored black, as was the ravine initially, indicating that they are now known. Each robot's sensors can view approximately 4 pixels away from the robot itself.

In Figure 3, the dark robot has reached the left bridge.

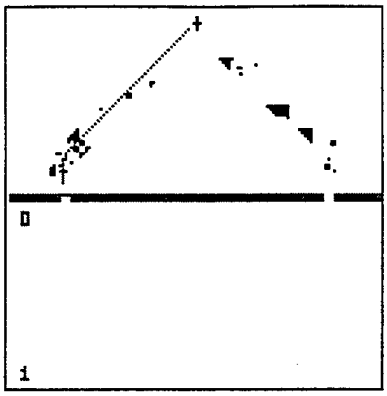


Figure 3: Simulation Step 2

However, the left doorway is now black, indicating that the robots sensors have observed the bridge and thus, that the planners are aware that the robot cannot move through these regions. The planner at this point realizes the difficulty and redefines each robot's goal selection to minimize the time to mission completion. The light robot will now go to the left goals, since it is closer to them than the dark robot, and the dark robot will go to the right goals.

Figure 4 shows that the light robot swapped the order of its goals since it went slightly out of the way while traveling to the upper left goal.This occurred because the swap reduced the total time for mission completion.

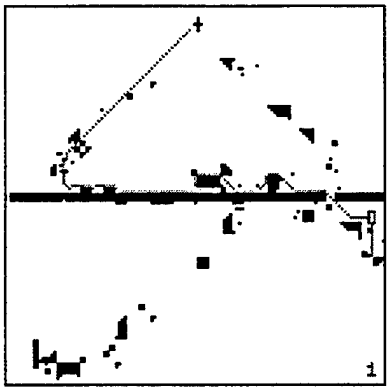


Figure 4: Simulation Step 3

Also, each robot has reached the first of its two goals and has started heading towards the second. Finally, in Figure 5, both robots returned to base, and the light robot picked a quicker route around the right side of the indicated obstacle as received obstacle information from dark robot during its less efficient traverse to the left

side.

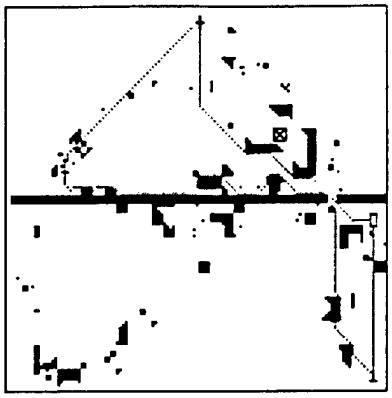


Figure 5: Simulation Step 4

By continuously comparing the cost of alternative mission, the Mission Planner ensured that each robot was always moving to the correct goal with respect to optimizing the total time to complete the mission. The Dynamic Planner ensured the path followed to that goal was optimal. Map information was shared between robots to minimize the time for repeated traverses of familiar areas. Each robot encountered many unknown objects in the environment, and navigated in an optimal manner to its current goal given its current knowledge of the world. The worst case cycle time of the mission planner was under one second on a Sparc10. All processing was performed on one computer -- whereas in a real scenario, one computer per robot (at a minimum) would be available.

If dynamic mission planning had not been performed, light would have reached the goals on the right and returned to the base quickly, leaving dark far behind, having to take much extra time to reach its goals (given that bridge was actually blocked.) Instead, the robots effectively cooperated through the Mission Planner, optimizing time to mission completion via reassignment of goals as permitted by the mission statement. Furthermore, since dynamic planning was also used when the robot was navigating toward it's single goal, it optimized computation performed by not requiring complete replanning of the paths to its goal each time an unexpected obstacle was encountered.

A more complex scenario involving three robots and six goal in shown in Figure 6. Randomly generated scenarios[1] of this sort were simulated 1000 times, both with the type of dynamic planner used in the first example, and with a static planer which never changed the ordering of the goals from the initial plan.

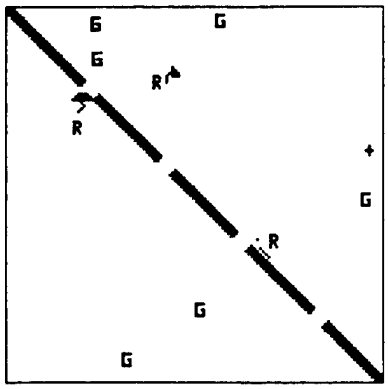


Figure 6: Advanced Scenario

1. The obstacles, number of open/closed bridges along the ravine, and location of the robots and goals were all randomized.

Dynamic mission planning was found to result in missions which were approximately 25% more efficient on average than those statically planned. On average, missions were changed (i.e. goal were reassigned) 5.2 times during their execution.

6. Conclusion and Future Work

This system, along with prior work in dynamic planning for outdoor navigators, has demonstrated that complex missions can be performed using a reasonable computational capacity via a straightforward and feasible expansion of existing planning and navigational frameworks. It is expected that this design will permit the execution and optimization of a large class of previously unattempted missions for mobile robots.

Given the ability of this type of planner to select millions of alternative possibilities efficiently, which permits real time optimization of mission plans, it is the intent of future work to implement a mission planner capable of understanding the full grammar discussed in Section 2. The system will have the ability to distribute computation, use local inter-robot collision avoidance schemes, and gracefully degrade optimality when presented with computationally intractable plans. The system will be demonstrated on two autonomous off-road mobile robots as well as further validated in simulation.

7. References

[1] Allen, J., et al. *Readings in Planning*. Morgan Kauffman Publishers. San Mateo, CA. 1990.

[2] Brooks, R.A. A Robust, Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*. 1986.

[3] Cameron, J.M., MacKenzie, D.C. *MissionLab: User Manual for Missionlab preliminary version 0.5*. Personal Communication. Dec., 1994.

[4] Feldman, J.A., Sproull, R.F. Decision Theory and Artificial Intelligence II: The Hungry Monkey. *Cognitive Science 1*. 1977.

[5] Fikes, R.E., Nilsson, N.J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, Vol. 2. 1971.

[6] Kaelbling, L.P. An Architecture for Intelligent Reactive Systems. See [1] pp.713-728.

[7] Kelly, A. *An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem*. Carnegie Mellon Robotics Institute Technical Report, CMU-RI-TR-95-33.

[8] Lacroix, S., et al. Autonomous Navigation in Outdoor Environment: Adaptive Approach and Experiment. *Proceedings ICRA*. 1995.

[9] Langer, D., et al. An Integrated System for Off-Road Navigation. *Proceedings ICRA*. 1994.

[10] Latombe, J.C. *Robot Motion Planning*. Kluwer Academic Publishers. 1991.

[11] Lawler, E.L., et al. *The Travelling Salesman Problem*. John Wiley & Sons. 1985.

[12] Le Pape, C. A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling. *Proceedings ICRA*. 1990.

[13] Mataric, M.J. Minimizing Complexity in Controlling a Mobile Robot Population. *Proceedings ICRA*. 1992.

[14] Parker, L. *Heterogeneous Multi-Robot Cooperation*. Ph.D. Thesis, MIT, Feb. 1994.

[15] Parsons, D., Canny, J. A Motion Planner for Multiple Mobile Robots. *Proceedings ICRA*. 1990.

[16] Rugg-Gunn, N., Cameron, S. A Formal Semantics for Multiple Vehicle Task and Motion Planning. *Proceedings ICRA*. 1994.

[17] Stentz, A. Optimal and Efficient Path Planning for Partially-Known Environments. *Proceedings ICRA*. 1994.

[18] Stentz, A. The Focussed D* Algorithm for Real-Time Replanning. *Proceedings IJCAI*. 1995.

[19] Stentz, A., Hebert, M. A Complete Navigation System for Goal Acquisition in Unknown Environments. *Autonomous Robots*, 2(2). 1995.

Chapter VI: Systems

Introduction

Several systems integrating map building, road following, range data, and path planning were developed. Those systems were demonstrated using the HMMWV in a natural environment. Systems building contributes to the development of UGV technology in several important ways. First, it allows us to evaluate and demonstrate the performance of the individual mobility components in the context of realistic missions. Second, building systems for different applications demonstrates the flexibility of our architectural tools. Finally, our systems were prototypes of the systems used in the UGV demos. Developing and demonstrating a series of systems permitted the early detection of problems and limitations.

We have included detailed descriptions of three integrated systems in this Chapter. In the first system, the vehicle travels on the road using ALVINN, avoiding obstacles using an earlier version of SMARTY and the ERIM laser range finder. The system computes the vehicle position by matching the positions of landmarks observed in the range images with their positions in a stored map. The Annotated Map system is used for updating and keeping track of vehicle position and for predicting landmark matches. In addition to landmarks, the system uses the observed road orientation to correct vehicle pose.

This first system demonstrates the ability to combine different mobility modules, in this case road-following and obstacle avoidance, using the DAMN arc arbitration module. It also illustrates the use of the Annotated Map system for integrating and controlling multiple modules. Finally, the uncertainty on sensor measurements is taken into account explicitly at every step of the system, including vehicle position, landmark positions and road orientation. The various sources of uncertainty are combined to yield optimal vehicle pose estimation using a Kalman filter.

A similar system was demonstrated at Lockheed Martin as part of UGV Demo B. That system did not include uncertainty representation but did use the Annotated Map system to switch between modes (road-following, obstacle-avoidance, teleoperation) and to update the parameters of the various modules at specific locations along the route.

In the second system focuses on obstacle avoidance and cross-country driving. The system navigates through natural terrain by combining the steering outputs of SMARTY with the output of a "goal seeking" module, which steers the vehicle to a goal point. The goal of this system is to test the obstacle-avoidance module over long distances. In the experiment described in the paper included in this Chapter, we report autonomous driving over one kilometer. This was achieved by placing intermediate goal points at intervals of approximately 100m. This experiment demonstrated the robustness of the obstacle detection and local mapping algorithms.

A limitation of this second system is that the vehicle is controlled from information based entirely on local obstacles. In particular, that system is unable to deal with situations such as cul-de-sacs and it does not follow an optimal path to the goal points. Furthermore, the addition of goal points may not be possible in a realistic mission in which a single goal might be preferable. For those reasons, we developed the third example system presented in this Chapter.

This system also uses SMARTY, but, instead of combining the SMARTY votes with a simple goal-seeking algorithm, it uses the D* dynamic route planner. Using D*, we can specify a single goal point arbitrary far from the starting point and lets the vehicle find the optimal path to the goal using the obstacle map discovered by SMARTY as it travels. The experiments reported in the third part of this chapter show that using D* enables the vehicle to travel over multi-kilometer paths, to discover its environment on the fly, and to update its route accordingly. In particular, the vehicle has to explore large areas in order to resolve situations in which its path is completely blocked by a cul-de-sac.

This system validates the performance of D* as predicted by the theoretical analysis of Chapter V. Also, it demonstrates the ability of the DAMN arbiter to combine steering recommendations from planning modules as well as from perception-based modules, such as obstacle avoidance. This is an important step since our behavior-based approach is based on the claim that arbitration of steering and speed is sufficient for building complex systems, including planning modules as well as low-level driving modules.

After demonstrating this system, a modified version of it was implemented and demonstrated at Lockheed Martin as part of UGV Demo C. The Demo C system was augmented with a map-sharing capability in which the internal D* obstacle map was transmitted from a first vehicle to a second vehicle, allowing the second vehicle to take a more direct route to the goal. This demonstration highlighted the accuracy of the SMARTY/D* obstacle map and the potential for multi-vehicle operation.

Both the CMU and Lockheed Martin versions of the third system use the SAUSAGES mission planning tool described in Chapter II to control the behaviors and to set their parameters.

Integrating Position Estimation and Perception for Autonomous Navigation

1. Introduction

Autonomous navigation of an outdoor mobile robot involves many tasks, including road following, obstacle avoidance, landmark recognition, path planning, and navigation. The test-bed robots, the Navlab and Navlab II, use a variety of sensors and sensor interpretation strategies for these tasks. Typical sensors include video cameras, for road tracking; a scanning laser rangefinder, for detecting obstacles and recognizing landmarks; sonars or giga-hertz radar for quick obstacle detection; shaft encoders and gyroscopes for position tracking. In this chapter, we describe one of our early integrated systems that used multiple sensing algorithms, incorporated information from a map, and used an explicit representation of the uncertainty on sensor and position measurements.

In the first systems, we typically used a single sensor, interpreted by a single module, to control the vehicle for a single task. Individual systems did road following, or path tracking, or obstacle avoidance. As the components became more sophisticated and more reliable, we have combined multiple sensors or multiple interpretation modules for a single sensor to build more capable systems. These systems avoid obstacles while driving along roads, or recognize landmarks while driving, or follow pre-planned paths while checking for obstacles.

Recently, the focus shifted to using maps in combination with more than one sensor interpretation module. The most ambitious missions combine road following, landmark recognition, obstacle detection, and inertial navigation to drive through an unmodified suburban setting, through intersections and along streets, stopping at a designated destination. We used several different strategies to keep track of a vehicle's position on the map: dead reckoning kept a running position estimate, perception for road following generated lateral corrections, and landmark recognition generated corrections along the direction of travel.

There were two related problems with our simple position update scheme. First, the corrections were made independently by each module, so the position estimate was completely determined by whichever module had made the most recent correction. Thus, if one of the corrections was slightly incorrect, it influenced the entire system. Second, the position estimates carried no measure of error bounds. The landmark recognition subsystem, in particular, needed predictions from navigation to focus the attention of the sensing and the processing. Without a measure of the error bounds for a particular position report, the landmark recognizer did not know how far in advance to begin looking, or how large an area to search.

The solution was to use filtered updates, rather than absolute updates. A very simple Kalman filter could easily give approximate bounds on vehicle position, and could be used to integrate new measurements with current estimates, instead of completely throwing out the old position. We have built and tested such a system.

One of the biggest challenges in building our system to filter position updates was calibrating the individual sensing modes. Section 2. describes our methods for estimating the error covariances for the laser scanner, dead reckoning, and road following. Another important part of the project was adapting our software structure to make the updates relatively time-independent, and to integrate position information with maps. Our approach to these problems is described in Section 3. The actual filter itself, presented in Section 4., is clean and simple. Finally, in Section 5. of this paper we discuss our experimental results.

2. Position Measurement and Error Modeling

2.1. Position and Covariances from the Laser Scanner and Object Matching

In this system, the ERIM sensor was used to measure 64 row by 256-column range images at a rate of two images per second. The range values are coded on 8 bits from 0 to 64 feet, yielding a range resolution of three inches. Raster scanning a scene is achieved by the motion of two mirrors synchronized so that the vertical and horizontal angular steps between consecutive pixels in the image are constant. The vertical and horizontal fields of view are 30 and 80 degrees, respectively. Details about the sensor and its applications may be found in [5][7][3]. We focus in this paper on building and using maps of objects detected from the range images, and to manipulate the uncertainty on the locations of the objects.

To detect objects in a range image, the algorithms introduced in [4] are used: First, The 3-D coordinates of each pixel from the range image are computed. Each point is stored in a cell of a horizontal discretized grid. The mean, maximum, and minimum elevations and the surface normal are computed for each cell of the grid and stored. Individual cells are classified as being part of an object based on the similarity of these parameters. Adjacent object cells are grouped into regions, each region corresponding to an individual object. Erroneous initial cell classification might occur, since the parameters are computed at each cell separately. This is corrected by retaining only those regions that are large enough. Spurious objects may still be detected because of noise in the data, for example. Those are eliminated using multiple images of a scene. This technique is simple and efficient and is appropriate for navigation in mild environments where there is a clear separation between terrain and objects.

2.2. Uncertainty Modeling

The uncertainty in range measurement may be modeled as a Gaussian distribution of standard deviation σ proportional to the square of the measured range, R. The proportionality factor K depends on the physical characteristics of the sensor, on the reflectivity of the material, ρ, and on the angle of incidence of the beam, α. Although K should vary linearly with ρ, we assume that ρ is nearly constant in our application. This assumption is justified by the fact that the objects used in the experiments reported in this paper were all of similar material. A more accurate model would make use of the intensity reflected by the target to adjust the uncertainty in range. With this simplification, we model the uncertainty as

$$\sigma_R = \frac{K' R^2}{\cos\alpha} \tag{1}$$

K' is computed empirically using the experiments described in [6].

This model is valid only when the uncertainty due to measurement noise dominates the uncertainty due to the sampling of the measurements. This condition is satisfied at the distances that we are considering, typically 5 to 15 meters.

In addition to range uncertainty, we have to take into account the uncertainty in the horizontal and vertical scanning angles, denoted by σ_φ and σ_θ respectively. Typical values of angular uncertainty are 0.1 degrees. Furthermore, we need to convert the uncertainty in (range,angles) space to uncertainty in the (x,y,z) Cartesian space. This can be done in two ways. In the first approach, we observe that the uncertainty on the full measurement $m = (\varphi, \theta, R)$ can be represented by a 3x3 covariance matrix, Σ. Assuming that range and angles are uncorrelated, the matrix is diagonal, the diagonal elements being σ_φ^2, σ_θ^2, and σ_R^2. We then observe that the Cartesian coordinates $p = (x,y,z)$ of a point are functions of range and angles, $p = F(m)$, where F is a transfor-

mation similar to the standard transformation from spherical to Cartesian coordinates. The covariance matrix representing the uncertainty on p is therefore $C=J\Sigma J^t$, where J is the Jacobian of the function F. The matrix J is a function of the scanning angles φ and θ only and can therefore be precomputed at every pixel position in the image.

In the second approach, a coordinate system (X,Y,Z) is defined such that Z is aligned with the direction of measurement, and X and Y are in the plane orthogonal to Z. The uncertainty is modeled by a diagonal covariance matrix W. The last element of W is σ_R^2, the first two elements are estimated from the constant uncertainty on scanning angles. The transformation between measurement reference frame (X,Y,Z) and sensor reference frame (x,y,z) is defined by a rotation U, therefore the covariance matrix is given by $C=UWU^t$. As in the first approach, the matrix U is a function of the scanning angles φ and θ only and can therefore be precomputed at every pixel position in the image.

Although the covariance matrix C is a full 3x3 matrix in general, we use only the 2x2 matrix that corresponds to the marginal distribution in x and y. This is because the vehicle model used in our experiments includes vehicle position and heading, but does not include elevation, roll, or pitch. To avoid clumsy notations, the notation C will refer to the 2x2 covariance matrix in the remainder of the section. The uncertainty on the position of a given object is also a 2x2 covariance matrix that is computed from the points detected on the object.

2.2.1. Matching Objects

The main tool for navigation that uses objects extracted from range images is the matching of objects detected in an image with objects detected in previous images. The problem can be stated as follows: given a current set of objects O at locations $O_i=(x_i,y_i)$, with covariance matrices C_i, and a new set O' at locations $O'_i=(x'_i,y'_i)$, with covariance matrices C'_i, and a transformation between the reference frames of O and O', find the most consistent set of matches between O and O'. The first step is to predict individual matches. The criterion for building the predictions is to declare that object O'_i is a possible match for O_j if UO'_i+T is within the 95% confidence ellipse defined by the covariance C_i, where U and T are the rotation and translation part of the translation between the two coordinate frames. This step may generate several possible matches for a given object. A search procedure evaluates the possible combinations of matches and retains the one that is the most consistent. It should be noted, however, that the search is fast because there are only a few ambiguous matches. The ambiguity is limited because of the small number of objects in the scene and because the estimate of the transformation (U,T) is accurate enough to avoid frequent mismatches. The output of the matching process is a set of matches (O_i, O'_j). Using those matches to reduce the uncertainty on the locations of the objects and to update the position estimate is the object of the next two sections.

2.2.2. Map Building

The goal of map building is to combine multiple observations into a consistent map of the environment. In our case, a map consists of a set of objects described by their locations and their associated uncertainty. Using the notations of the previous section, O is the set of objects currently stored in the map, and O' is the set of objects detected in the new image. Matches (O_i,O'_j) are computed as described in the previous Section. For each of those matches, the current estimate of the position object O_i is updated using a standard linear Kalman filter. More precisely, the updated position O''_i of O_i is given by:

$$O''_i = O_i + K(UO'_j + T - O_i) \qquad K = C_i(C_i + UC'_jU^t)^{-1} \tag{2}$$

In this expression, (U,T) is the current estimate of the transformation between map and vehicle position at the time the image was taken, and K is the usual Kalman gain. The covariance matrix is updated by:

$$C''_i = (C_i^{-1} + UC'^{-1}_i U^t)^{-1} \tag{3}$$

Using this type of filtering is beneficial only if several views of each map object are available, in which case the accuracy of object location estimate increases dramatically. In a typical map building run, images are collected as fast as possible thus yielding significant overlap between images. A typical object is visible in five to ten images, thus allowing the filtering process to be effective. A limitation of this approach to map building is the computation time required to process this large amount of data. As a result, map building must be performed off-line or at low vehicle speed.

In addition to refining object locations, map building includes a mechanism for eliminating objects erroneously detected in an image. The detection of those spurious objects may be caused by noise in the input data or mismatches. In addition, moving objects must be eliminated because the system can handle only static environments. This problem is essentially an outlier removal problem and thus cannot be handled by a linear filtering technique such as the Kalman filter. The outlier removal mechanism that we implemented is based on the fact that it is easy to predict which map objects should be visible in an image, given the approximate position at which the image is taken and a model of the field of view of the sensor. We define the confidence of a map object as the number of times it is successfully found in a sequence of images versus the number of times it is predicted to appear in the same sequence. A real object will have a confidence close to one, while an outlier will have a very small confidence since it occurs in only one image in most cases. A moving object would be detected in consecutive range images, but each instance of the object would be too far from the location detected in the previous images to yield successful consistent matches. Thus, the confidence is much smaller than one for moving objects as well. Erroneous objects are therefore eliminated by removing map objects with a confidence lower than a threshold. This confidence mechanism is another reason for using a large number of closely spaced images because the confidence of an object is meaningful only if it has been predicted in a large enough number of frames. Figure 1 shows this tracking of objects in ERIM images taken at different vehicle positions.

2.2.3. Position Estimation

Position estimation from range data is the dual of the map building. As before, it involves matching a set O of map objects and a set O' of objects from a new range image. This time, however, the goal is to refine the current estimate of vehicle position given by a rotation U and a translation T defined by the position of the vehicle (x_v,y_v) and its heading θ. V denotes the vector (x_v,y_v,θ), and C_v denotes the 3x3 covariance matrix of V. The discrepancy between the current position estimate and the estimate given by the matches between map and image objects must be estimated first. ΔV denotes the discrepancy vector $(\Delta x_v,\Delta y_v,\Delta\theta)$, and $C_{\Delta v}$ denotes its covariance. The true rotation between map and current observation is given by $U'=U+\Delta U$, where ΔU is the error in rotation. Assuming that the angular error is small, we can make the approximation $\Delta U=\Delta\theta J$, where J is the derivative of the rotation matrix U with respect to the rotation angle θ. The true translation is given by $T'=T+\Delta T$, where $\Delta T=(\Delta x_v,\Delta y_v)$. Using the same notations as before for the objects, we have: $O_i=U'O'_j+T'$. Therefore, replacing U' by $U+\Delta U$ and T' by $T+\Delta T$, we obtain: $O_i-UO'_j-T=\Delta U\,O'_j+\Delta T$. Denoting the left-hand side of the equation by P_i and applying the small angle approximation, the relation becomes: $P_i=\Delta\theta J+\Delta T$. Since the right-hand side is a linear function of ΔV, the relation can be written as $P_i=H\Delta V$, where H is a 2x3 matrix. Since this is a linear relation between the measurement P_i and the estimated variable ΔV, we can directly apply the formalism of the linear Kalman filter. More precisely, a new estimate of the discrepancy $\Delta V'$ can be computed from the current estimate ΔV, given a new match, using the relations:

$$\Delta V' = \Delta V + K(P_i - H\Delta V) \qquad K = C_{\Delta V}H^t(C_{P_i} + HC_{\Delta V}H^t)^{-1}$$
$$C'_{\Delta V} = (C_{\Delta V} + H^t C_{P_i} H)^{-1} \tag{4}$$

In this equation, C_{Pi} is the covariance matrix of P_i which can be easily computed from C_i, C'_j, and C_v.

The position update could be approached through a different route by observing that the relation $O_i=UO'_j+T$ is of the form $F(O_i,O'_j,V)=0$, where F is a non-linear function of V that incorporates the trigonometric functions

of θ that appear in U. Viewing (O_i, O'_j) as the measurement, and V as the estimated variable, we can directly apply the Extended Kalman Filter formulas [2] to get a new estimate of V. In both cases, the underlying assumption is that the angular and positional errors are small enough to justify local linearization.

In practice, the position update is activated by the map whenever objects are predicted to appear in the current field of view of the sensor based on the estimated vehicle position. The error vector ΔV is initialized to 0 when the position update module is activated and is updated every time a new match is found between map and image objects. The module processes images until a sufficient number of matches is found, or until a maximum number of images, typically ten, is reached. The first stopping criterion is needed because a minimum number of matches is required for the final ΔV to be meaningful. The second criterion is needed because only the final estimate is sent to the other modules, which might have to wait for a long time if there were no limit on the number of images processed, thus causing significant delays in the navigation system.

The output of this Kalman filter, ΔV and $C_{\Delta v}$, are position corrections based on landmark observations. They are converted to estimates of vehicle position P_m and its covariance C_m by adding the vehicle's position V and by dropping the rotation terms. The P_m and C_m terms are then fed to the Navigator module for its internal Kalman filter for overall vehicle position estimation, as described in Section 3.

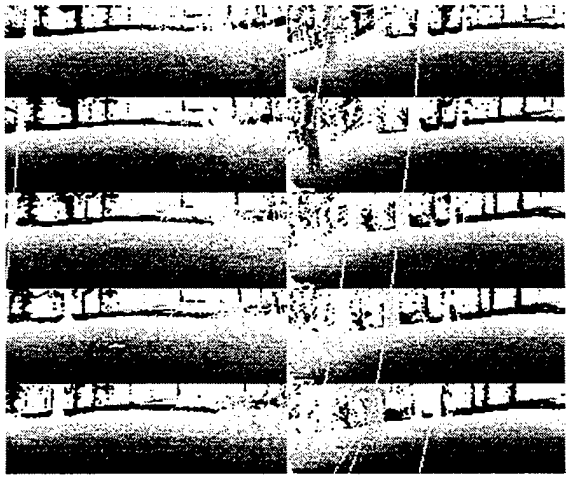


Figure 1: Fourteen ERIM laser range finder images. Dark pixels are close to the sensor and light pixels are farther away. Corresponding objects are linked together across images.

2.3. Dead Reckoning

2.3.1. Measuring model for odometry

The odometer reports the vehicle's distance traveled. Since this data is vital to vehicle positioning and sensor modeling, a good model for the odometry system is essential.

There are two primary factors contributing to odometry data errors: terrain type and vehicle lateral acceleration (i.e. turning radius and velocity). The terrain type has a dramatic effect on the accuracy of an odometer due to slippage. For example, it is very easy to slip while traveling on loose gravel or mud versus asphalt. Slippage can also occur as a result of varying lateral acceleration. Driving though sharp curves at a constant velocity will cause more slippage than driving straight. For this study, we confined our driving to asphalt, so we modeled lateral acceleration slip but not slips induced by terrain type.

The odometer of our vehicle consists of a shaft encoder reporting data from the differential of the rear axle. In order to model this encoder data, experiments were conducted in a very large, flat, asphalt parking lot in order to keep the terrain type uniform and similar to typical situations. For each trial of the experiment, the vehicle was driven in a circle while the vehicle speed was kept as constant as possible (3-5 m.p.h.). The vehicle's real time controller kept the steering wheel at a constant position for each trial. Between successive runs, the steering wheel position was changed in small intervals in order to collect data across the full range of steering directions. At each steering wheel position, the vehicle was driven in a circle and the true circumference of the circle was measured by a fifth wheel measurement device pulled behind the vehicle. In cases where the circle was too large for the available space, the vehicle was driven on smaller portions of a circle (half, quarter, etc.). The data from the fifth wheel was recorded and averaged if multiple data was taken and was used to compute the true distance traveled by the vehicle's origin (center of the rear axle). The number of encoder counts received for each trial was recorded as well as the steering wheel position. Figure 2 shows the result of this experiment.

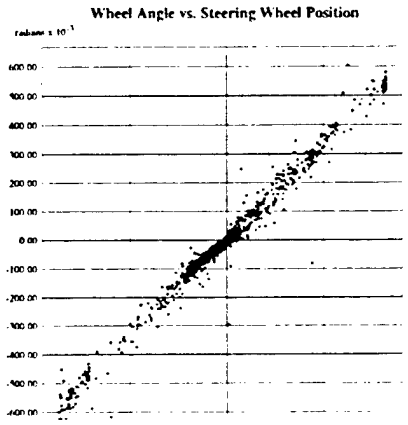


Figure 2: Noise in distance traveled and its dependence on vehicle steering. Vertical axis is calibration of distance traveled. Horizontal axis is steering wheel position from left turn to right turn.

The extremes of the x axis represent the sharpest turning radii of the vehicle where one would expect the most slippage to occur. This is exactly what occurs as illustrated in Figure 2 by the longer distances traveled per encoder count at the extremes. Also, as expected, the slippage diminishes as the radii become less sharp. Unfortunately, contrary to our expectations, the data are not symmetrical about zero curvature, and it is difficult to fit curves or use tables for estimating an accurate odometry calibration function. One reason for this asymmetry may be uneven loading of the vehicle, causing more slip while making right turns. This and other hypotheses must be further investigated by more data collection in the future. The nature of the current data made it impossible to accurately model the dependence of the odometry accuracy on turning radius. Instead, we simply used the mean and the variance of the data in our position estimation algorithm. Additional data was collected by driving the vehicle on a straight line between two carefully surveyed points twenty times in order to have a better estimate of the mean and the variance. The mean for distance traveled was 0.38885 mm/ encoder count with a standard deviation of 0.0072785.

The vehicle's lateral position is also susceptible to slip. We discovered the extent of this slippage by measuring the lateral position error at the termination point of the completed circles. As expected, the error reduced with decreasing curvature. More data is required for determination of the exact relationship of the error to curvature. The mean (0) and the variance (0.51% of distance traveled) of this data is currently used for error estimation.

2.3.2. Measuring model for steering angle

The turning radius of the vehicle is another crucial variable in position estimation. Estimating the turning radius requires accurate measurement of the effective front wheel angle α. The simplest way to measure α is to measure the position of the steering wheel. This requires a calibration function mapping the steering wheel position to α. To determine this function, the distance traveled was sampled from the odometer Δs and changes in heading $\Delta\varphi$ were recorded from a gyroscope at short time intervals as the vehicle was driven on a wide variety of paths at different speeds. The ratio of Δs to is the sensed radius of curvature.

Provided that the interval is short enough, we can assume that the steering wheel position is essentially constant during this interval. Therefore, we can associate the measured radius of curvature with the particular steering wheel position. The sensed curvature was stored in a table indexed by the steering wheel position and vehicle speeds. For a better approximation of the steering wheel position, the steering encoder positions reported at the beginning and end of the time interval were averaged and used to index the curvature table. Different measured radii for the same steering encoder position were averaged as well. Figure 3 illustrates the data in these tables.

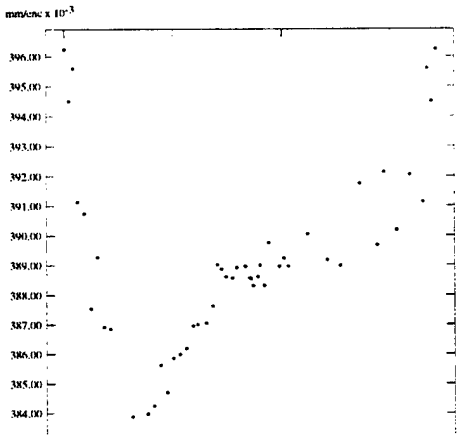


Figure 3: Wheel angle vs. steering wheel position.

There are several possible reasons for the uncertainty in the estimated curvature from the steering wheel position. Tire slippage and deformation cause under/oversteering which varies the turning radius. Distortion in the sidewalls of the tires is considered to be a major contributor to these errors since it causes the tire treads to move in a different direction from the wheel rims. More detailed analysis of these factors and effects of vehicle dynamics are necessary to discover other sources for this uncertainty in the curvature estimation. Position estimation using the steering wheel and odometry data was quite accurate for distances less than twenty meters in typical situations. For longer distances, the heading error was significant and prompted the usage of more accurate sensors for vehicle heading. To reset the heading error, a gyroscope was used to measure vehicle heading and it was found to be extremely fast and accurate. Currently, the data from the gyroscope in conjunction with the odometry is used for position estimation.

2.4. Neural Network Road Following

The ALVINN neural network-based vision system is used for the road-following component of the system. The output of the ALVINN network is a steering direction from which the lateral position of the road with

respect to the vehicle may be computed. In order to refine the vehicle's estimated position using the output of the road-following system, it must have an accurate model of uncertainty associated with the road follower's performance. Since the road-following module is used to make lateral corrections to the estimated vehicle position relative to the road, the important aspect of its performance to quantify is how closely the road follower keeps the vehicle to the center of the road (or the center of the appropriate lane on a multi-lane road). More specifically, we need to measure the mean and variance of the road follower's lateral error with respect to the road.

Measuring the road-follower's accuracy is a two-step process. The first step involves precisely determining the position of the road center. This measurement process is difficult on the single lane road used as the primary test site for this research, since broken pavement and fallen leaves frequently made estimating the exact location of road edges difficult. The technique chosen to ensure maximum measurement accuracy is manually locating the road edges and marking the point midway between them as the road center. To avoid disturbing the vision system with spurious image features, the road center points are marked with paint that contrasts only slightly with the pavement. We delineate the road center points once per meter over a 140-meter stretch of the test road, which included a straight stretch and turns to the left and right.

The second step in quantifying the driving accuracy of the road-following module is to measure how closely the vision system kept the vehicle to the road center while driving autonomously. Because of the low contrast between the pavement and the road center markings, the only feasible technique for determining how close to the center the road follower keeps the vehicle is to measure it manually. Accurate real time measurement of the vehicle's relative position every meter proved impractical, since the vehicle's usual speed is at least two meters per second. So instead, we have designed an apparatus that allows the vehicle to leave a trail that can be carefully measured after the test run. Specifically, we immersed one end of a narrow tube in a bucket of water, and attached the other end to the origin of the vehicle's coordinate system, in the center of the rear axle. Using this apparatus as a siphon, the vehicle drips water along the precise path it follows over the test road. By measuring the deviation of this drip trail from the road center markings, we can estimate the mean and variance of the vision system's road-following error. Since the trail is only water, it evaporates quickly. This prevents the trail from interfering with trails made on subsequent runs.

With the road follower driving at 5 miles per hour along the 140 meter test section of road, the vehicle's mean distance from the road center is 1.62 cm, with a variance of 52.49. These figures compare favorably with the same measurements made when a person drives the vehicle. Under human control, the vehicle's mean distance from the road center is 4.02 cm, with a variance of 30.03. It appears that the human driver, while more consistent than the vision system, had a inaccurate estimate of the vehicle's centerline, and therefore drove slightly right of the road center. Studies of human driving performance have found similar steady state errors and variances in vehicle lateral position. Blaauw [1] found consistent displacements of up to 7 cm were not uncommon when people drove on highways. On the same task, Blaauw reports standard deviations in lateral error ranging from 16.6 to 36.4 depending on the individual.

3. Annotated Maps

Once the individual components are built and calibrated, we still need a software framework to tie all the pieces into a coherent and useful system. We use the EDDIE architectural toolkit to combine information from the road follower, from the landmark recognizer, from dead-reckoning, from the map, and from human planning in order to perform an autonomous mission. EDDIE consists of a communications library, which sets up direct communication links between processes, a database library, which maintains a local "list" of objects in the map, and a set of resources. These resources are servers that clients can talk with via the communications library. The two most significant resources are the Annotated Maps module [9], which is used as both a means to efficiently query the object database and as a positional alarm clock, and the Navigator module, which is used to maintain the best estimate of the vehicle's position on the map.

The object database contains all the information about the world. The map can contain the following types of

objects:

- **Generic objects:** Discrete circular or polygonal objects, e.g., trees, mailboxes.
- **Road points:** Road points are linked together to form roads.
- **Intersections:** Intersections link different roads together.
- **Meta objects:** Meta objects are used primarily for global information about roads. For example, a road's meta object might contain the name of the street.
- **Alarms:** These are artificial objects placed in the map for control purposes.

Each object in the database can be annotated with extra information that individual clients in the database have to interpret.

The Annotated Maps module takes the list of object descriptions and forms a two-dimensional grid so it can answer queries about the object database more efficiently. For example, the object matcher has to know what objects are in its field of view. To find this out, it could look through the object database, checking for objects that overlap the field of view polygon, but this would be very inefficient for a large database. Instead, the recognizer can send the polygon to the Annotated Maps module, which can use the grid to quickly find the objects within the polygon.

The Annotated Maps module is not just a passive geometric database; it is actively involved in the control of the system. The alarm objects are placed in the map by a human user to plan the scenario. Alarms are conceptual objects in the map, and can be lines, circles, or regions. Each alarm is annotated with a list of client modules to notify and the information to send to each when the alarm is triggered. When the Annotated Map manager notices that the vehicle is crossing an alarm on the map, it sends the information to the pertinent modules. The map manager does not interpret the alarm annotations: that is up to the client modules.

Alarms can be thought of as positionally based production rules. Instead of using perception based production rules such as, "If A is observed, then perform action B", an Annotated Map based system has rules of the form, "If location A is reached, then perform action B." Thus, we reduce the problem of making high-level decisions from the difficult task of perceiving and reacting to external events to the relatively simple task of monitoring and updating the vehicle's position.

The first step in building an Annotated Map is collecting geometric information about the environment. We build our maps by driving the vehicle over roads and linking the road segments together at intersections. Once a road is built, the human user can create a "meta object" for that road, which describes the whole road. In this system, we annotate a road's meta object with a short description or label and with the absolute orientation of the vehicle at the start of the road. At the same time, a laser range finder is used to record the positions of landmarks such as mailboxes, trees, and telephone poles. The landmark collector also annotates each object it finds with the two dimensional covariance of the object's position, in addition to its location and description, which can be used later by the object matcher. After the map building phase, a human planner draws "trigger lines" across the road at various locations. For example, the person knows that when approaching an intersection, the vehicle should slow down, and so chooses to put the trigger line at the approach to an intersection. The trigger line goes across the road at that point, and is annotated with a string of bits that represents the new speed of the vehicle. During the run, when the vehicle crosses the trigger line, the map manager sends the string of bits to a module that interprets the information and slows the vehicle to the desired speed. In our current system, alarms are interpreted as commands, but there is no predefined "correct" way for a module to react to an alarm. Depending on its content, an alarm could also be interpreted as a walk-up call, or even as simply advice.

The EDDIE toolkit glues together our system, with the database providing the information, with the Navigator fusing position estimation's to pinpoint position, and with the Annotated Maps module providing the symbolic

information and control knowledge necessary for a fully autonomous mission.

Because position information is so critical to an Annotated Map system, the Navigator resource module uses multiple knowledge sources to determine the vehicle's current location in the map. The Navigator uses the Kalman filter techniques described elsewhere in this paper to combine positioning information from our inertial navigation system, from the road follower, and from the landmark matcher to refine the vehicle's map position. The Navigator sends updates of this position to all modules that need global map positioning.

4. Filtered Position Updates

4.1. Corrections

In the current implementation, the Navigator maintains a correction transform rather than the actual position. The vehicle controller provides positioning information completely independently of the map. This positioning information is derived from gyros and dead reckoning, and is in an arbitrary frame of reference that we call the controller coordinates. Client modules that do not need the map can use positioning information in controller coordinates. For example, a road-following vision system might be interested only in the motion of the vehicle between frames, since it does not need to know the absolute position of the vehicle on the map, so it can directly use the controller position to get relative motion. Other modules, which need to use the map, need the transform T_{world} which converts controller coordinates to map coordinates. This T_{world} transform is maintained by the Navigator. Thus, a module that needs to know the current position of the vehicle on the map must first acquire the current T_{world}, then query the controller for the current vehicle position in controller coordinates, then apply T_{world} to convert to map coordinates.

There are several advantages of this system. First, the controller coordinates are never updated, so modules do not have to worry about jumps in relative locations of nearby objects that are stored in controller coordinates. In other schemes, when a position correction happens all objects stored in the current local coordinate system must be updated. Second, the position corrections that do occur (when the T_{world} is updated) can occur in a separate, possibly slower process (the Navigator), without interrupting the real-time loops of the controller. This decomposes our system logically into a fast system running in local coordinates, and a separate set of often slower processes that need to refer to the map. Besides being a good design tool, this is also a practical help in doing the updates. Time is no longer as important a quantity. Directly updating the controller's position would require changing the position estimate in real time, while the controller is still doing dead reckoning. In our scheme, instead, the Navigator's T_{world} can be updated with no real-time concern, since it is not rapidly changing. Also, filtering the relatively stable quantity T_{world} is relatively easy since it will typically require only small changes. This also means that the compounding operation, which increases positional uncertainty during dead reckoning between landmark fixes, is also simplified. The mean of T_{world} stays constant during compounding, and only the positional covariance C_p must be changed.

4.2. Filter Design

To estimate the vehicle's position as accurately as possible, odometry data, gyroscope heading data, ERIM data, and road information are used in conjunction with previously recorded maps to estimate vehicle position and uncertainty. The vehicle heading is measured by a commercial gyroscope, which employs its own Kalman filters to incorporate compass and gyroscope data to produce exceptionally accurate heading data. This gyroscope was made by the Litton corporation for aiming guns accurately. For this reason, the filter formulation is only concerned with the vehicle's x and y coordinates, assuming perfect heading (theta) estimation from the commercial gyroscope.

The Navigator uses the real-time controller's position estimation derived from the odometry and heading data

as a basis for an internal transform developed from ERIM and map information. The Kalman filter is a black box to the Navigator, which internally maintains the position uncertainty. The Navigator invokes two operations in this formulation, the compounding and the merging. Figure 4 shows the covariance compounding and merging equations.

Compounding Equations	Kalman Filter Equations
$C_p' = J\begin{bmatrix} C_p & 0 \\ 0 & C_D \end{bmatrix} J^t$	$C_p' = (C_p^{-1} + C_M^{-1})^{-1}$ $P' = P + K(P_M - P)$
$C_p = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{bmatrix}$ $C_D = \begin{bmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_d^2 \end{bmatrix}$	$P = \begin{bmatrix} x \\ y \end{bmatrix}$
$J = \begin{bmatrix} 1 & 0 & \cos\theta & -\sin\theta \\ 0 & 1 & \sin\theta & \cos\theta \end{bmatrix}$	$K = C_p(C_p + C_M)^{-1}$

Figure 4: Kalman filter equations.

The compounding operation is invoked at regular distance intervals of 0.5 meters. The covariance matrix for dead reckoning, C_D, contains the variance of lateral error σ_l and distance traveled σ_d. These variances are assumed independent. σ_l and σ_d are updated using the variances previously determined from odometry data and road follower performance. The new position covariance matrix C_p' is computed by using the Jacobian J to compound C_D and the old C_p.

The merging operation is invoked when another module provides an estimate of the vehicle position. In our current system, these additional estimates occur when the object matcher finds a new vehicle position from landmarks or when the road follower provides a steering command. The Navigator is given a vehicle position P_m measured by one of the perception systems and corresponding covariance matrix C_m. P_m and C_m are merged with the current estimates of position P and uncertainty P' by the Kalman filter. This merging operation updates the covariance, C_p', and returns the updated position P'. The Navigator uses the new map position P' and the controller position (x_c, y_c) to generate a new T_{world}.

The landmark matcher directly gives the Navigator a new estimated vehicle position P_m and its covariance C_m. The Kalman filter module directly uses these values, plus the old position estimate P, as the inputs to its filter.

To update P_m and C_m from road information, the Navigator follows a similar procedure. First, the Navigator uses the current transform T_{world} to estimate vehicle position P. It then projects P onto the road to find the measured vehicle position P_m, since the road follower is assumed to keep the vehicle on the road center. Road updates can move the vehicle position perpendicularly to the road but not along the road. The Navigator therefore forms a covariance matrix C_m oriented along the road, with the road-follower's error estimate perpendicular to the road and an essentially infinite variance along the road. As in landmark matching, the P_m estimate, the C_m covariance, and the P prior position estimate are passed to the Kalman filter module to generate a new P' and C_p. The new P' is used by the Navigator to build its new T_{world}.

5. Experiments and Results

5.1. Results

The system was tested on a twisting park trail near campus. A human driver drove up the road to map it while the object detector found landmarks, such as large trees. The object detector annotated each landmark with its observed covariance in x and y. After we built the map, we put one trigger line in it that tells the landmark matcher to start looking for landmarks during a run. One such map is shown in Figure 5.

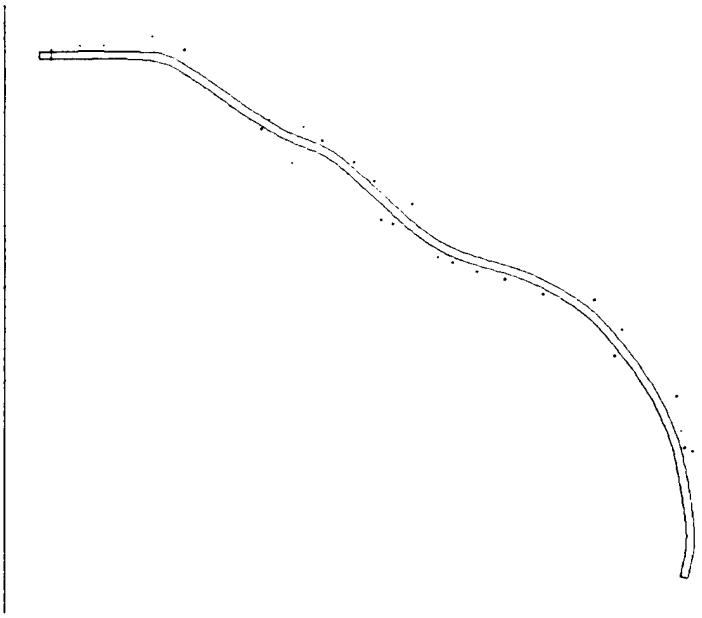


Figure 5: Map of test area showing the road and trees.

To start a run, the driver drives the vehicle onto the road, and the operator then estimates the vehicle's initial position. The initial orientation is calculated by taking the difference between the angle currently reported by the gyros and the gyro angle annotated in the nearest road and adding that difference to the map angle annotated in the nearest road. The amount of drift in the gyros was small enough that this method of orienting the vehicle was much more accurate than just guessing the vehicles orientation. We estimated the initial uncertainty of the vehicles position as 3 meters in x and y. Figure 6 shows the uncertainty ellipse of the vehicle in the middle of the run after being shortened along the direction of the road by landmark matching and after being thinned by updates from the road follower.

One interesting result is that making lateral road corrections on a curving road can correct an error in position along the road. This is not immediately obvious intuitively since an individual road correction gives no information about the position of the vehicle along the road, only information about the lateral position of the vehicle on the road. This effect was first noticed by observing the changes in uncertainty ellipses during a run. In Figure 7, a vehicle has a circular uncertainty at position **a**, and after a road update at position **b**, the uncertainty ellipse becomes a needle in the direction of the road. Imagine that the vehicle travels through the corner. At position **c** it has an uncertainty ellipse very similar to at position **b**, but after another road update at position d, the ellipse shrinks drastically. From local road updates, we are vastly more confident about the position of the vehicle along the road at position **d** than at position **a**. This implies that the apparently difficult problem of using road curvatures as landmarks is solved by simply making local corrections to the vehicle's position per-

pendicular to the road.

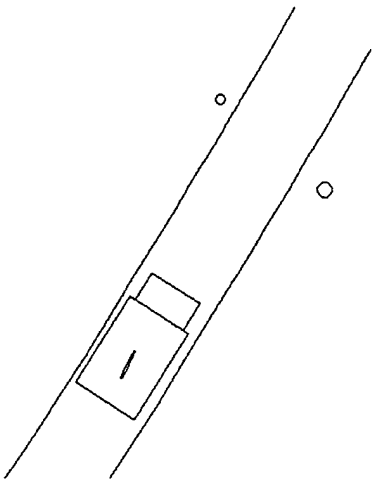


Figure 6: Vehicle position and uncertainty during a run.

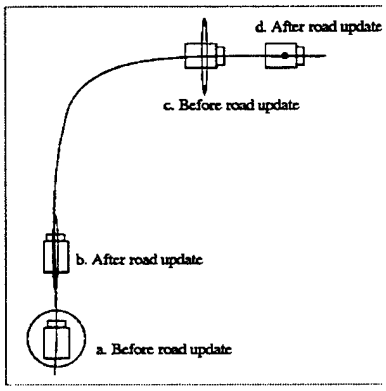


Figure 7: Effects of road updates on position uncertainty.

This effect was evident in the experiments.The vehicle was deliberately started in the wrong place in the map, displaced along the road by 4 meters. The landmark matcher could not match landmarks with this large an error, so it failed initially. After going through a curve with just road updates, the position in the map was close enough to the position in the real world that the landmark matcher was able to match sensed objects with map objects and further refine the position.

Are there cases in which the road updates confuse the position along the road? Unfortunately, yes. The autonomous road follower drives slightly differently than the human driver who made the map. For instance, the road follower may cut a corner slightly more than the human driver. Our model assumes that the vehicle is in the center of the road, even though it is not. A series of very slightly erroneous corrections going through a corner will result in an error in the distance along the map. Figure 8 illustrates the problem. This problem could be addressed by having the road follower be the driver when making the maps, or by increasing the compounding error in the direction of travel when going through a curve, or by simply not performing road updates when the vehicle is going through a curve.

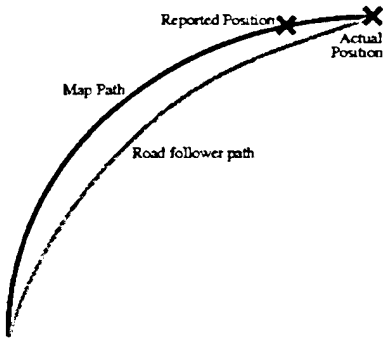


Figure 8: Errors caused by road updates during inaccurate driving through a turn.

Performance was measured qualitatively over a series of runs with the vehicle, and some preliminary quantitative measurements from system performance were derived. Qualitatively, the most important criterion is that the landmark matcher should be able to find landmarks within its expected error. We found that to be the true in nearly all cases. Quantitatively, at the end of typical runs of 350 meters, the position updates generated by the matcher were usually between 5 to 10 cm.

6. Future Work

There are several areas still open for investigation.

Curves and matching. Since the road follower provides position updates only perpendicular to the road, we rely on knowing the road direction fairly accurately. If the vehicle is in a tight curve, and has significant positional error along the road, it will also have significant uncertainty in the direction along which road updates should take place. We need to model and account for that error, or to not perform road updates while driving on tight curves.

Map errors. Our error formulations does not account for errors in road position on the map. We do record the covariance of the estimate of landmark positions, but that is relative to an assumed perfect vehicle position on the road. A fairly accurate approximation is that the map has the same error characteristics as the vehicle driving, so treating the map as perfect and doubling the error estimates during driving should work. Better estimates of map error will become more important as we continue to work on building more accurate maps.

Correlated errors. We treat our errors as zero-mean, uncorrelated, with a Gaussian distribution. In fact, errors in certain systems like the road follower tend to be correlated in time: if the vehicle is 10 cm off at one instant, it is unlikely to be 10 cm off in the opposite direction in the next instant. This undoubtedly limits our performance.

Dead reckoning errors. We know that errors in dead-reckoning are linked to radius of curvature. We believe that lateral and longitudinal errors are also linked. The current data sets do not give us a meaningful basis for representing those relationships. The net effect is to generate another source of correlated errors. If the vehicle drives along a long curve with constant radius, the mean distance traveled will have a small bias from the mean over all possible directions of travel.

Measuring system performance. It is comparatively easy to measure system repeatability, by making several runs over the same route and recording the end position of the vehicle. It is much harder to measure absolute accuracy. The current test course is lined with trees, so there is no clear line of sight along which to use a the-

odolite. We will either have to subcontract a surveyor to make a complete map, or run in a less interesting test site just to make measurements.

Other environments. The tests described in this paper involve road following with relatively discrete landmarks, such as trees. The precision of our dead-reckoning will degrade when we make cross-country runs on dirt and mud, and the precision of landmark matching will be less when we are trying to recognize less structured features, such as piles of rocks.

Bibliography

[1] G. J. Blaauw. Driving Experience and Task Demands in Simulator and Instrumented Car: A Validation Study. *Human Factors*, 24:473--486. 1982.

[2] A. Gelb. *Applied Optimal Estimation.* MIT Press. 1974.

[3] M. Hebert. Building and Navigating Maps of Road Scenes Using Active Range and Reflectance Data. In Charles E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab.* Kluwer Academic Publishers. 1990.

[4] M. Hebert. Building Object and Terrain Representations for an Autonomous Vehicle. In *Proceedings American Control Conference.* 1991.

[5] M. Hebert and T. Kanade. Outdoor Scene Analysis Using Range Data. In *Proceedings of International Conference on Robotics and Automation.* IEEE Computer Society. 1986.

[6] M. Hebert and E. Krotkov. Imaging Laser Radars. In *Proc. IROS* . 1991.

[7] M. Hebert, I. Kweon, and T Kanade. 3-D Vision Techniques for Autonomous Vehicles. In Charles E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab.* Kluwer Academic Publishers, 1990.

[8] D. Pomerleau, J. Gowdy, and C. Thorpe. Combining Artificial Neural Networks and Symbolic Processing for Autonomous Robot Guidance. *Journal of Engineering Applications of Artificial Intelligence*, 1991.

[9] C. Thorpe and J. Gowdy. Annotated Maps for Autonomous Land Vehicles. Pittsburgh PA. 1990.

An Integrated System for Autonomous Off-Road Navigation

1. Introduction

Autonomous navigation missions through unmapped open terrain are critical in many applications of outdoor mobile robots. To successfully complete such missions, a mobile robot system needs to be equipped with reliable perception and navigation systems capable of sensing the environment, of building environment models, and of planning safe paths through the terrain. In that respect, autonomous cross-country navigation imposes two special challenges in the design of the perception system. First, the perception must be able to deal with very rugged terrain. Second, the perception system must be able to reliably process a large number of data sets over a long period of time.

Several approaches have been proposed to address these problems. Autonomous traverse of rugged outdoor terrain has been demonstrated as part of the ALV [10] and UGV [9] projects. JPL's Robby used stereo vision [8] as the basis of its perception system and has been demonstrated over a 100 m traverse in outdoor terrain. Other efforts include: The VAP project, which is also based on stereo vision [2]; the PANORAMA Esprit project [14]; the MIT rovers, which rely on simple sensing modalities [1]. Most of these perception systems use range images, from active ranging sensors or passive stereo, and they build a map of the terrain around or in front of the vehicle. The planning systems use the maps to generate trajectories. The main questions in building such systems are: What should be in the map, and when should the map be computed?

In this paper, we argue that relatively simple methods of obstacle detection and local map building are sufficient for cross-country navigation. Furthermore, when used as input to a reactive planner, the vehicle is capable of safely traveling at significantly faster speeds than would be possible with a system that planned an optimal path through a detailed, high-resolution terrain map. Moreover, we argue that an accurate map is not necessary because the vehicle can safely traverse relatively large variations of terrain surface.

For these reasons, we propose an approach based on "early evaluation of traversability" in which the output of the perception system is a set of untraversable terrain regions used by a planning module to drive the vehicle. The system relies on "early evaluation" because the perception module classifies regions of the terrain as traversable or untraversable as soon as a new image is taken. As we will show, early traversability evaluation allows for a more reactive approach to planning in which steering directions and speed updates are generated rapidly and in which the vehicle can respond to dangerous situations in a more robust and more timely manner.

The goal of this paper is to present and discuss the performance of the overall system. Detailed descriptions of its components may be found in [7] for the local map module, and in [11] for the planning component.

2. System Overview

To illustrate our approach, we will describe a set of perception and navigation modules that constitute the core of a cross-country navigation system. The goal of this system is to enable the vehicle to travel through unmapped rugged terrain at moderate speeds, typically 2 to 3 meters per second. We arranged the system modules in a self-contained navigation system, which we demonstrated on a 1-kilometer path through unmapped open terrain. In the next sections, we will use this result to illustrate our approach and to discuss the system performance and the implementation details of each module.

The support vehicle is the HMWVV and the sensor is the Erim laser range finder, which acquires 64x256 range images at 2 Hz. An estimate of vehicle position is available at all times by combining readings from an INS system and from encoders.

In the current system, the output of the perception system is a set of untraversable terrain regions which is maintained by a local map manager and is used by a planning module to drive the vehicle. The location of the untraversable regions with respect to the vehicle is maintained by a local map manager. Untraversable regions are terrain features, such as high slopes, ditches, or tall objects that would endanger the vehicle.

In the remainder of the paper, we first describe the performance of the perception and navigation system on an example, then, we describe the three components of the system (Figure 1), perception, local map management, and planning, with an emphasis on the perception component. Finally, we discuss the pros and cons of the early evaluation approach.

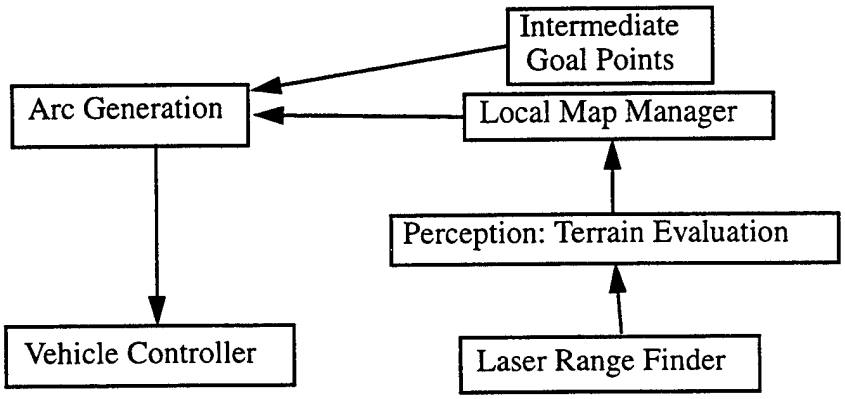


Figure 1: Architecture of the navigation system.

3. Autonomous Navigation: An Experiment

Figure 2 to Figure 4 show a typical run of the perception and navigation system. Figure 2 shows the environment used in this experiment, which includes hills, rocks, and ditches. The white line superimposed on the image of the terrain shows the approximate path of the vehicle through this environment. The path was drawn manually for illustrative purpose.

Figure 3 shows the actual path recorded during the experiment projected on the average ground plane. In addition to the path, Figure 3 shows the obstacle regions as black dots and the intermediate goal points as small circles. In this example, the vehicle completed a 1-kilometer loop without manual intervention at an average speed of 2 m/s. The input to the system was a set of 10 waypoints separated by about 100 meters on average. Except for the waypoints, the system does not have any previous knowledge of the terrain. Local navigation is performed by computing steering directions based on the locations of untraversable regions in the terrain found in the range images. An estimated 800 images were processed during this particular run.

Figure 4 shows close-ups of three sections of the loop of Figure 3. The black lines show the approximate paths followed by the vehicle in these three sections. Figure 4 (b) shows the elevation map obtained by pasting together the images taken along the paths. In each figure, the grey polygons are the projections of the fields of view on the ground, the curved grey line is the path of the vehicle on the ground, and the grey dots indicate locations at which images were taken. In this case, the images are separated by approximately 2 meters. The paths shown in Figure 4 (b) are the actual paths followed by the vehicle. It is important to note that these maps are included for display purposes only and that the combined elevation maps are not actually used in the system. Finally, Figure 4 (c) shows three views of the local map which is maintained at all time around the vehicle. The squares corresponds to 40x40 cm patches of terrain classified as untraversable regions or obstacles. These local maps are computed from the positions shown in Figure 4 (a) and Figure 4 (b) by the white arrows. The trajectories are planned using this compact representation.

All the displays, timings, and results used in the rest of the paper are from the site of Figure 2, although they

may have been collected over different runs of the system. The system was configured to run on three SparcII workstations. Each workstation was dedicated to one of the three components of the system: perception, planning, and local map management. Communications between processes were established through Ethernet. Controller and sensor interface have their own dedicated processors running a real-time operating system, while the other processes are Unix-based.

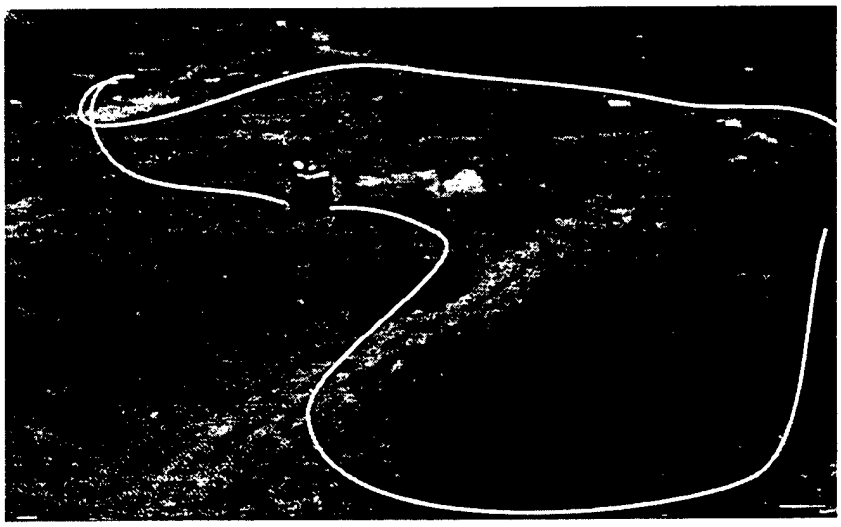


Figure 2: View of terrain and approximate path.

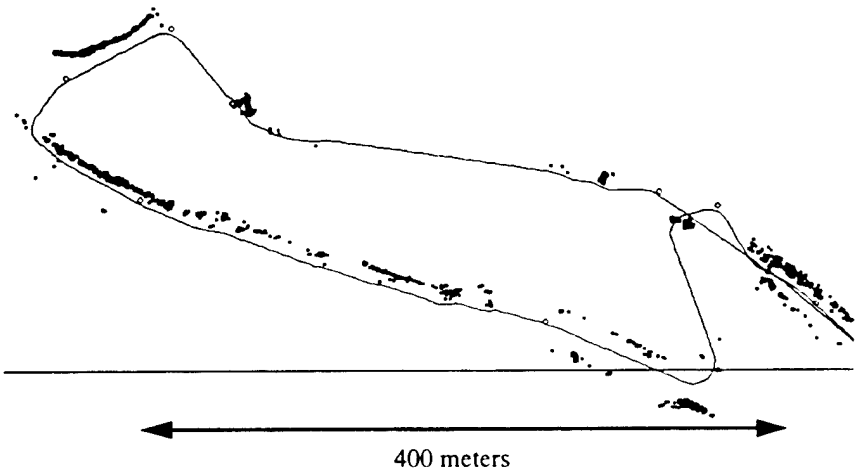
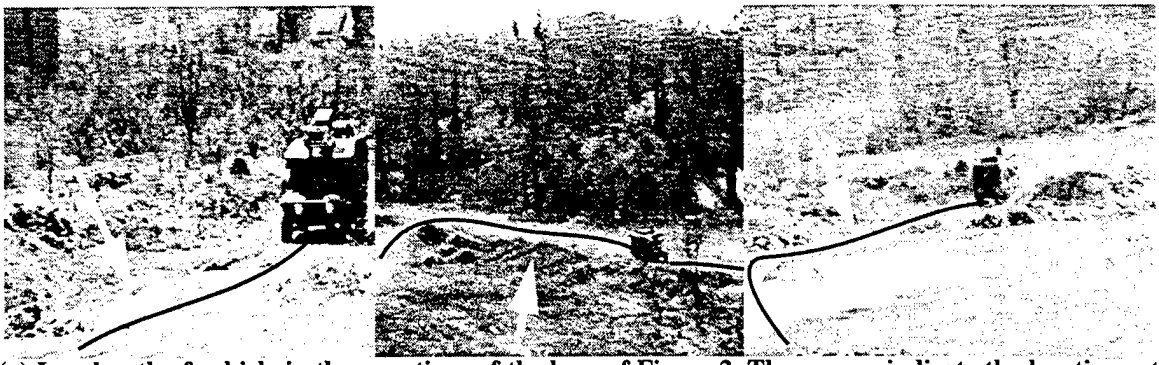
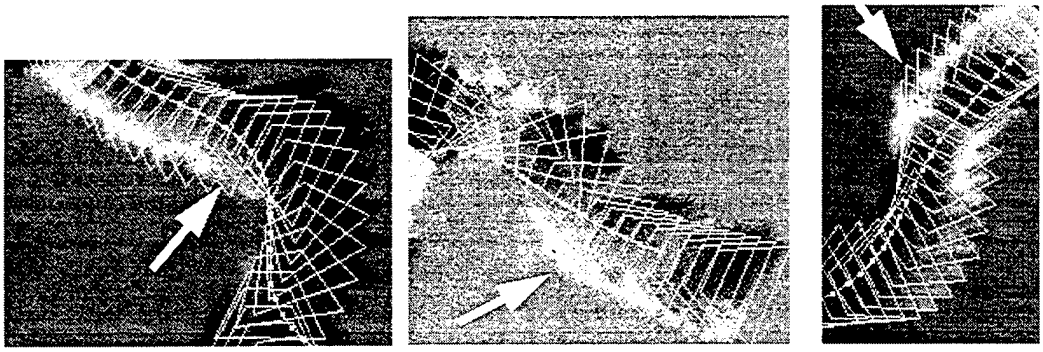


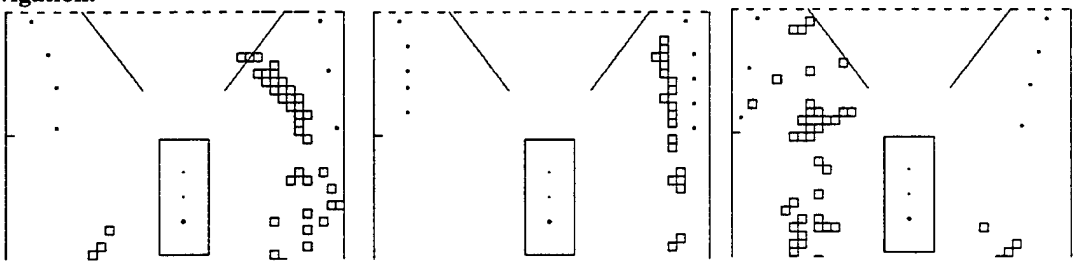
Figure 3: Exact path of vehicle; the obstacle regions are shown as black dots; the intermediate goal points are shown as small circles.



(a) Local path of vehicle in three sections of the loop of Figure 2. The arrows indicate the locations at which the local maps are displayed in (c) below.



(b) Display of the sensed terrain as elevation maps for the sections shown in (a) . The polygons indicate the projection of the field of view of the sensor on the ground. The grey line shows the path followed by the vehicle in this section. The grey dots show the positions at which the images were taken. The arrows are placed at the same locations as in (a). This elevation map is for display purposes; it is not used in the navigation.



(c) Display of the local traversability map at the locations marked by arrows in (a) and (b). Only the portion of the map in the immediate vicinity of the vehicle is displayed here. The vehicle is depicted by a rectangle. The untraversable regions are shown as squares.

Figure 4: Detailed views of three sections of the loop of Figure 2

4. Perception

The range image processing module takes a single image as input and outputs a list of regions that are untraversable. After filtering the input image, the module computes the (x,y,z) location of every pixel in the range image in a coordinate system relative to the vehicle's current position. The transformation from sensor to vehicle takes into account the orientation of the vehicle read from an INS system. The points are then mapped into a discrete grid on the (x,y) plane. Each cell of the grid contains the list of the coordinates of the points that fall

within the bounds of the cell. The size of a cell in the current system is 20 cm. This number depends on the angular resolution of the sensor, in this case 0.5^{o}, and on the size of terrain features that need to be detected. The terrain classification is first performed in every cell individually. The criteria used for the classification are the height variation of the terrain within the cell, the orientation of the vector normal to the path of terrain contained in the cell, and the presence of a discontinuity of elevation in the cell. To avoid frequent erroneous classification, the first two criteria are evaluated only if the number of points in the cell is large enough. In practice, a minimum of five points per cell is used.

This range image processing algorithm has several important properties. First, it does not build a complete, high-resolution map of the terrain, which would require interpolating between data points, an expensive operation. Instead, the algorithm evaluates only the terrain for which there is data. Second, the algorithm processes each image individually without explicitly merging terrain data from consecutive images. Instead, it relegates the task of maintaining a local map of untraversable regions to a separate local map module. The importance of this is that the local map module deals only with a few data items, the cells classified as untraversable, instead of with raw terrain data. As a result, maintaining the local map is simpler and more efficient. Because of these two features, range image processing is typically on the order of 200ms on a conventional Sparc II workstation.

It is clear that the range image processing module may miss untraversable regions of the terrain because the terrain is evaluated only where data is present in the image and because the data may be too sparse to provide complete coverage of the terrain at long range. However, because of the processing speed, a region that is missed in a given image will become visible in subsequent images quickly enough for the vehicle to take appropriate action. Although this problem effectively reduces the maximum detection range of the perception system, we argue that the other possible solutions would reduce the maximum range even further and would introduce additional problems.

The most obvious alternative is to merge data from a few images before committing to a terrain classification. This solution effectively reduces the maximum detection range because the system has to wait until enough overlapping images are taken before a terrain region is evaluated. In addition, merging images is in itself a difficult problem because it requires precise knowledge of the transformation between images. In particular, even a small error in rotation angles between two images may introduce enough discrepancy between the corresponding elevation terrain maps to create artificial obstacles at the interface between the two maps. (We refer the reader to [5] for a more quantitative description of this problem.) Therefore, it is preferable to not merge images explicitly and to rely on fast processing to compensate for the sparsity of the data. In practice, 30cm terrain features are detected at a range of 10 meters from the sensor when vehicle speed is on average 2m/s. By comparison, the maximum range of the scanner is 18 meters with a distance between pixels of about one meter at that range.

It would be advantageous to vary the field of view of the range sensor dynamically as a function of speed and turning radius. Our range sensor does not have this capability at this time so the parameters are tuned for a nominal maximum speed instead of being adjusted dynamically.

5. Local Map Management

The purpose of the local map module is to maintain a list of the untraversable cells in a region around the vehicle. The current system uses Ganesha [7] as the local map module. In this system, the active map extends from 0 to 20 meters in front of the vehicle and 10 meters on both sides. This module is general purpose in that it can take input from an arbitrary number of sensor modules and it does not have any knowledge of the algorithms used in the sensor processing modules.

The core of Ganesha is a single loop in which the module first gets obstacle cells from the perception modules, and then places them in the local map using the position of the vehicle at the time the sensor data was processed (Figure 5). The sensing position has to be used in this last step because of the latency between the time

a new image is taken, and the time the corresponding cells are received by the map module, typically on the order of 600ms.

At the end of each loop, the current position of the vehicle is read and the coordinates of all the cells in the map with respect to the vehicle are recomputed. Cells that fall outside the bounds of the map are discarded. Finally, Ganesha sends the list of currently active cells in its map to the planning system whenever the information is requested.

Because the map module deals only with a small number of terrain cells instead of with a complete model, the map update is rapid. In practice, the update rate can be as fast as 50 ms on a SparcII workstation. Because of the fast update rate, this approach is very effective in maintaining an up-to-date local map at all times.

One advantage of Ganesha's design is that it does not need to know the details of the sensing part of the system because it uses only information from early terrain classification. In fact, the only sensor-specific information known to the map module is the sensor's field of view, which is used for checking for consistency of terrain cells between images as described below.

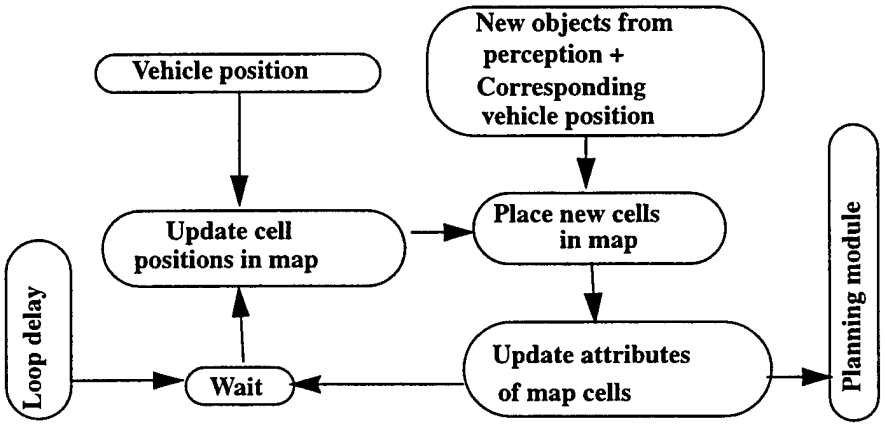


Figure 5: Main loop in Ganesha.

In this design of the navigation system, the local map and planning modules do not have access to the original sensor data and therefore cannot correct possible errors in the output of the perception. In particular, a region which is mistakenly classified as traversable will never be reclassified because the local map module cannot go back to the original data to verify the status of the region. It is therefore important to use conservative values for the detection parameters in order to ensure that all the untraversable regions of the terrain are classified as such.

The drawback of this approach is that the perception module may generate terrain regions that are incorrectly classified. Because the perception processes images individually without explicitly building maps, it cannot detect that the classification is inconsistent with previous observations. This problem is solved by the map maintainer, which does maintain a history of the observations. Specifically, an untraversable map cell which is not consistent across images, is discarded from the local map if it is not reported by the perception module as untraversable in the next overlapping images. Because the terrain classification is fast compared to the speed of the vehicle, many overlapping images are taken during a relatively short interval of distance travelled. As a result, an erroneous cell is deleted before the vehicle starts altering its path significantly to avoid it.

6. Path Planning

The third component of the system is the Distributed Architecture for Mobile Navigation (DAMN), which generates commanded steering radius and velocity with a high update rate [1]. The current system uses both an

obstacle avoidance behavior and a goal-seeking behavior. The obstacle avoidance behavior computes the votes based on the distribution of untraversable terrain cells around the vehicle as reported by the local map module. Arcs that steer the vehicle away from the untraversable regions are assigned a high vote, while arcs that would cause the vehicle to travel through a forbidden region are given a low vote. Every arc is evaluated by checking the intersection of each obstacle cell with an expanded version of the vehicle along the arc. A safety distance of 0.3 m around the vehicle is used in the evaluation. If an arc intersects a cell at a distance less than a pre-set minimum distance, then the arc is assigned a vote of -1; if an arc intersects a cell at a distance greater than a maximum distance, the vote is set to 1. In between the min and max distances, the vote is set to an intermediate value in a continuous manner.

The second behavior, goal seeking, assigns higher votes to arcs that steer the vehicle toward intermediate goal points. This second behavior ensures that the overall path of the vehicle follows the desired global trajectory. The votes are between -1 and +1. A set of 15 possible arcs is used in the experiment reported in Section 3.. The last module of the trajectory planner is an arbiter that combines the votes from the two behaviors and sends the arc with the highest vote to the vehicle controller.

This implementation of goal seeking is the simplest one that would allow us to demonstrate the system over a significant distance. More sophisticated modules, e.g., the global route planner D* [13], can be integrated in the system. The only requirement is that the higher-level planning modules be able to generates arrays of votes of the format required by DAMN at regular interval.

The command arbitration module combines the votes from the two behaviors and issues a new driving command every 100ms. The set of possible arcs used for the experiments is represented by the 15 turn radii shown in Figure 6. This set of radii is sufficient for the type of environment in which the vehicle travels. The goal points are on average 100 meters apart and the goal-seeking behavior switches goals whenever it comes within 8 meters of the current target goal point. The weights are 1.0 for the obstacle avoidance behavior and 0.25 for the seek goal behavior. After normalization, these are effectively 0.8 and 0.2, respectively. The absolute values are not important, only the relative weights. The obstacle weight is 4 times larger, since that is much more important than seeking the goal at any given moment.

Because the trajectory planner generates only local arcs based on compact local information, i.e., the obstacle cells, it has a high update rate and allows for rapid correction of small errors due to system delays or isolated perception errors.

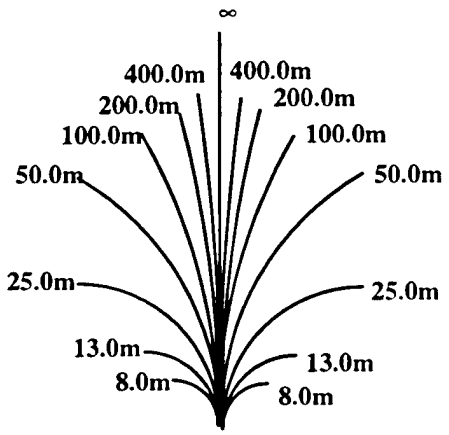


Figure 6: Set of possible turning radii in DAMN for the cross-country navigation system.

7. Combining Obstacle Detection and Goal Seeking: An Example

Figure 7 and Figure 8 illustrate more precisely the way DAMN combines outputs from different behaviors. Figure 7 shows the path of the vehicle, shown as a black line, around a set of obstacle points, shown as black dots. In this example, the vehicle was controlled by the steering commands issued by the arbiter, which is receiving votes from two behaviors, the obstacle avoidance behavior described above, and a heading behavior which forces the vehicle to follow a constant heading. The maximum vehicle speed was 2m/s in this experiment. The set of fifteen turn radii shown in Figure 6 was used in this example. The votes are displayed as shaded pixels with the brightest pixels corresponding to the highest possible vote of 1.0 and the black pixels corresponding to the lowest possible vote of -1.0. In addition to the raw votes from the arbiter, the figure shows an enhanced version of the arbiter's distribution of votes. The enhanced version is obtained by normalizing the pixel values using the effective min and max values for every time slot instead of the extreme values, -1.0 and 1.0. The enhanced arbiter is included to facilitate the visual interpretation of the vote distribution but it is not used for driving the vehicle. The commanded turning radius is computed from the maximum of the vote distribution of the arbiter as described above.

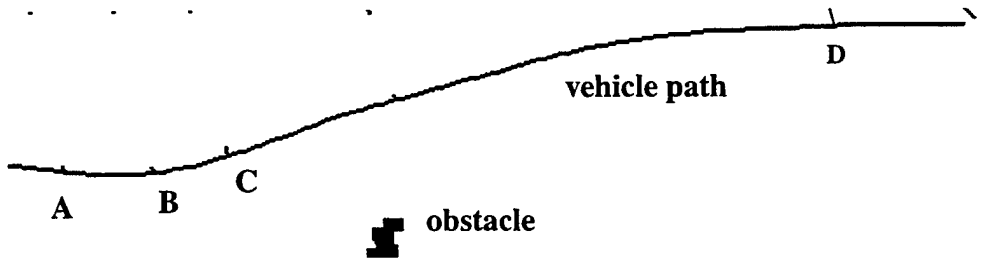


Figure 7: Example vehicle path around an obstacle.

Figure 8 shows the distribution of votes at five points of interest along the path, indicated by capital letters in Figure 7. Each graph depicts the votes issued for each turn choice by the obstacle avoidance and heading behaviors, as well as the weighted sum of these votes as computed by the arbiter. The horizontal axis of each graph shows the possible turn radius choices encoded from -8 meters for hard left to +8 meters for hard right.

The sequence of five points selected in Figure 7 is typical of a path around a single obstacle. At point *A*, the obstacle is first reported and the obstacle-avoidance behavior generates high votes for the left turns to go around the obstacle and inhibits the right turns with negative votes, as shown by the solid line in the graph. At the same time, the heading behavior's vote distribution is relatively flat around the straight direction since the vehicle is currently headed in the desired direction; this is shown by the dashed line in the graph. Because of the small relative weight of the heading behavior, the combined vote distribution in the arbiter, shown as a thicker solid line in the graph, is dominated by the votes received from the obstacle avoidance behavior; a left turn is therefore commanded. At point *B*, the obstacle is still close to the vehicle and the votes distributions are similar to the ones at *A*, thus maintaining the vehicle to the left of the obstacle. At point *C*, the obstacle avoidance behavior is still voting in favor of a sharp left turn, but the votes for the softer left turns are not as low as they were at *A* or *B*, since the vehicle is now clearing the obstacles. At the same time, the heading behavior is starting to shift its votes towards turning right in order to bring the vehicle back to the target heading. The summed votes are still at a maximum for a left turn because of the greater weight of the obstacle-avoidance behavior's votes, and so the arbiter continues to steer the vehicle well clear of the obstacles.

By the time the vehicle has reached point *D*, it is just passing the obstacles, so that the obstacle avoidance behavior is now only disallowing extreme right or left turns because of the field of view constraints. The heading behavior is now able to have more influence over the chosen turn direction, and a right turn is now executed

so that the desired vehicle heading is restored. Note that between points *C* and *D*, there is a transition period where the turn radius chosen by the arbiter is neither the hard left favored by the obstacle-avoidance behavior nor the medium right favored by the heading behavior. Instead, as turns other than hard left become acceptable to the obstacle-avoidance behavior, and as the distribution of votes from the heading behavior shift further to the right, the vehicle is commanded to make ever softer left turns, and eventually turns to the right. As can be seen in the graph of the commanded turn radius in Figure 8, rather than abruptly switching modes from avoiding obstacles to following a heading, the transition proceeds smoothly and steadily as the situation gradually changes. Finally, at point *E*, the vehicle has completely cleared the obstacles and they are no longer in the local map, so that the votes from the obstacle avoidance behavior are mostly +1. The heading behavior now dominates completely and a right turn is commanded. Through the interaction of the avoid obstacles and heading behaviors, the vehicle was thus able to successfully circumvent an obstacle and regain its desired heading. This simple experiment shows that the arbiter combines the preferred commands in an efficient and natural way

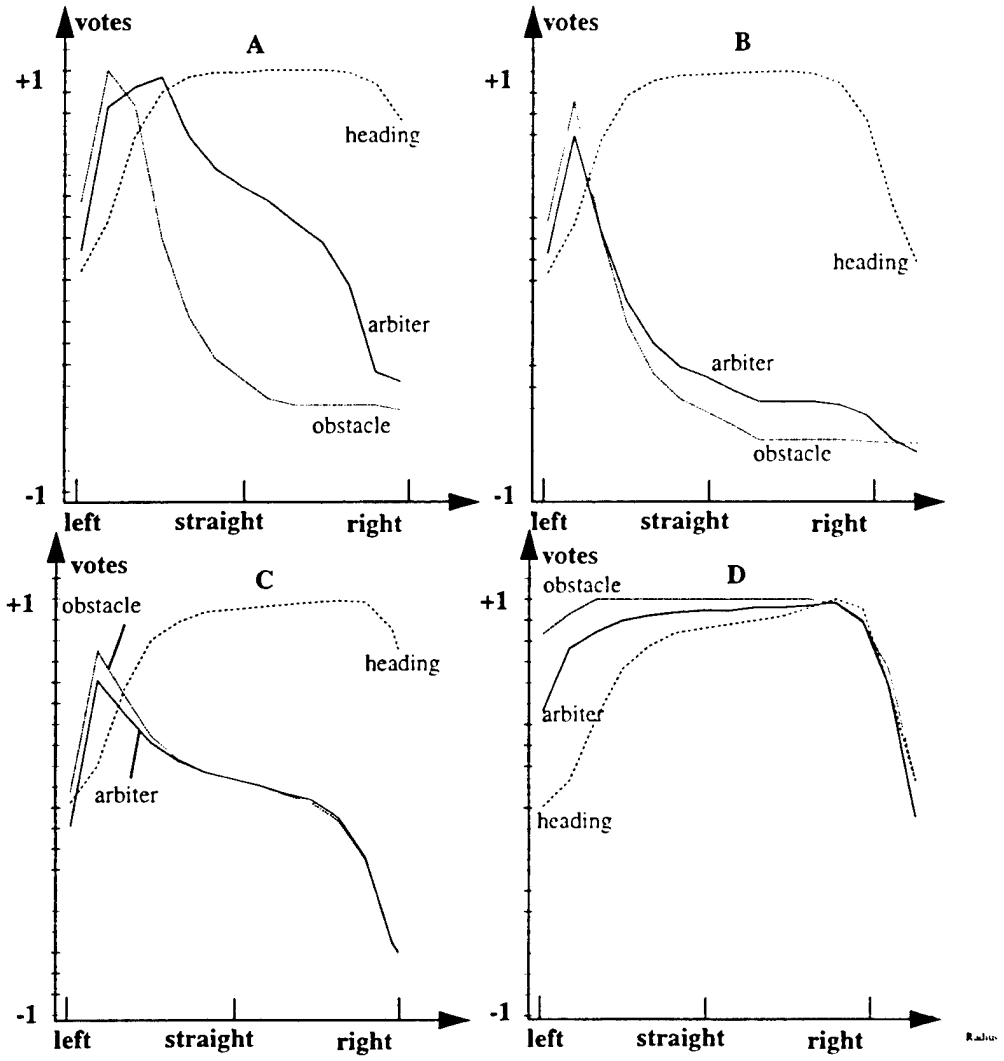


Figure 8: Distribution of votes in the arbiter (thick line), obstacle avoidance (thin line) and heading (dotted line) behaviors.

8. System Performance and Future Work

In summary, early evaluation of terrain traversability allows us to achieve continuous motion at moderate speeds by reducing the amount of computation required by the perception system, simplifying local map management and path planning, hiding the details of sensing from all the modules except perception, and avoiding the problems caused by merging multiple terrain maps using inaccurate position estimates. The drawback of this approach is that an error in the perception system cannot be corrected later in the system because only the perception module has access to the sensor data. This problem is eliminated by using a fast reactive path planner, an error-correction algorithm in the local map manager, and a simple perception algorithm with fast cycle time relative to vehicle speed, both of which allow the system to correct quickly for occasional perception errors.

In the current system, the vehicle can tolerate terrain discontinuities of 20cm at a speed of 3 m/s. With a range resolution of 7cm and an angular accuracy of 0.5^{o}, such a discontinuity can be detected in time to avoid it with an arc of radius greater than the minimum turning radius of 7.5 m, assuming a 2Hz image acquisition rate and an additional 0.5 seconds latency in the system. Although recent hardware improvements have raised the maximum range to 40m, the acquisition rate, maximum range, and resolution of the sensor are the numbers that set hard limits on the speed.

Another limitation of the system is the image-by-image nature of the perception module. Specifically, objects are reported to the appropriate behavior only after the entire image is processed. As a result, an object close to the vehicle is reported at the same time as objects at the maximum distance in the image. We have implemented a different version of the perception module, which processes the range data scanline per scanline and reports new obstacles every time a new scanline is processed. This perception module, SMARTY, is described in [3].

The other major factor in overall system performance is the distributed and asynchronous nature of the system. In particular, the message traffic between perception, Ganesha, and the behaviors may become heavy enough to introduce significant delays in the system.

This delay, on the order of 0.5 seconds, is small enough to permit reliable navigation at relatively low speeds. However, the situation worsens when a more efficient scanline-by-scanline processing of the range data is used, to the point where the system cannot keep up with the volume of data shipped between modules. A solution is to combine the range processing, the local map, and the corresponding behavior into a single module. In doing this, we retain the basic functionality of early traversability evaluation and of reactive planning while eliminating the network communication problems. Preliminary results on combining line processing and single module operation are also reported in [3].

Bibliography

[1] R. Brooks and A. Flynn. Fast, Cheap, and Out of Control: A Robot Invasion of the Solar System. *J. British Interplanetary Society*, 42(10):478-485, 1989.

[2] G. Giralt and L. Boissier. The French Planetary Rover VAP: Concept and Current Developments. In *Proc. IEEE Intl. Workshop on Intelligent Robots and Systems*, pp. 1391-1398, Raleigh, 1992.

[3] M. Hebert. Pixel-Based Range Image Processing for Autonomous Driving. *Proc. IEEE Robotics and Automation.* 1994.

[4] M. Hebert, E. Krotkov. 3D Measurements from Imaging Laser Radars. *Image and Vision Computing 10(3).* 1992.

[5] A. Kelly, T. Stentz, M. Hebert. Terrain Map Building for Fast Navigation on Rugged Outdoor Terrain. In *Proc. of the SPIE Conference on Mobile Robots.* 1992.

[6] I.S. Kweon. *Modeling Rugged Terrain by Mobile Robots with Multiple Sensors*. Ph.D. thesis. Robotics Institute, Carnegie Mellon University. 1991.

[7] D. Langer and C. Thorpe. Sonar-Based Outdoor Vehicle Navigation and Collision Avoidance. In *Proceedings IROS '92*. 1992.

[8] L. H. Matthies. Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time Implementation. *International Journal of Computer Vision*. 8:1. 1992.

[9] E. Mettala. Reconnaissance, Surveillance and Target Acquisition Research for the Unmanned Ground Vehicle Program. In *Proceedings Image Understanding Workshop*, Washington. D.C. 1993.

[10] K. Olin, D.Y. Tseng. Autonomous Cross-Country Navigation. *IEEE Expert*, 6(4). 1991.

[11] D.W. Payton, J.K. Rosenblatt, D.M. Keirsey. Plan Guided Reaction. *IEEE Transactions on Systems Man and Cybernetics*, 20(6). 1990.

[12] J.K. Rosenblatt and D.W. Payton. A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control. in *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*. Washington DC. 1989.

[13] T. Stentz. Optimal and Efficient Path Planning for Partially-Known Environments. *Proceedings IEEE Robotics and Automation*. 1994.

[14] T. Van Bogaert, P. Lemoine, F. Vacherand, S. Do. Obstacle Avoidance in Panorama Esprit II Project. *Proceedings IFAC Workshop on Intelligent Autonomous Vehicles*. Southampton. 1993.

A Navigation System for Goal Acquisition in Unknown Environments

1. Introduction

Autonomous outdoor navigators have a number of uses, including planetary exploration, construction site work, mining, military reconnaissance, and hazardous waste remediation. These tasks require a mobile robot to move between points in its environment in order to transport cargo, deploy sensors, or rendezvous with other robots or equipment. The problem is complicated in environments that are unstructured and unknown. In such cases, the robot can be equipped with one or more sensors to measure the location of obstacles, locate itself within the environment, and check for hazards. By sweeping the terrain for obstacles, recording its progress through the environment, and building a map of sensed areas, the navigator can find the goal position if a path exists, even if it has no prior knowledge of the environment.

A number of systems have been developed to address this scenario. We limit our discussion to those systems tested on a real robot in outdoor terrain. Outdoor robots operating in expansive, unstructured environments pose new problems, since the optimal global routes can be complicated enough that simple replanning approaches are not feasible for real-time operation. The bulk of the work in outdoor navigation has centered on local navigation, that is, avoiding obstacles [1][2][3][5][7][14][17][20][27][28][31] or following roads [4][8][15][25]. The global navigation capabilities of these systems have been limited to tracking a pre-planned trajectory, wandering, maintaining a constant heading, or following a specific type of terrain feature. Other systems with global navigation components typically operate by planning a global route based on initial map information and then replanning as needed when the sensors discover an obstruction [9][10][21].

These approaches are insufficient if there is no initial map of the environment or if the environment does not exhibit coarse, global structure (e.g., a small network of routes or channels). It is possible to replan a new global trajectory from scratch for each new piece of sensor data, but in cluttered environments the sensors can detect new information almost continuously, thus precluding real-time operation. Furthermore, sensor noise and robot position error can lead to the detection of phantom obstacles and the incorrect location of obstacles, flooding the global navigator with more, and sometimes erroneous, data.

We have developed a complete navigation system that solves these problems. The system is capable of driving an HMMWV from an initial position to a goal position. It may have a prior map of the environment or no map at all.Figure 1 shows the results of an actual run on the robot. The dimensions of the area are 500 x 500 meters. The robot began at the position labelled *S* and moved to the goal location at *G*. Initially, the robot assumed the world to be devoid of obstacles and moved toward the goal. Twice a second, the perception system reported the locations of obstacles detected by the sensor. Each time, the robot steered to miss the obstacles and replanned an optimal global trajectory to the goal. The robot was able to drive at approximately 2 meters per second. The robot's trajectory is shown in black, the detected obstacles in dark grey, and a safety zone around the obstacles in light grey.

This example is typical of the type of mission that we want to achieve and of the experiments reported in this paper. First, we desire to have optimal global path selection. The traverse shown in the figure is clearly not the shortest path from *S* to *G* once the entire obstacle map is known, but it is optimal in the sense that, at every point along the traverse, the robot is following an optimal path to the goal given all obstacle information known in aggregate at that point. Second, we desire to have real-time performance, even in large environments. Thus, we need to carry out experiments over long distances, typically greater than one kilometer.

For practical reasons, we make some simplifications in order to carry out our experiments. First, the goal point *G* is physically marked in the environment. We know that the goal has been reached successfully by checking that the robot has reached the marked position. The coordinates of *G* are the only input to the system at the

beginning of the experiment. More realistic scenarios would call for marking the goal with a specific landmark recognizable by a vision system, but this would add complexity without contributing to the validation of our planning and perception algorithms. Second, the laser rangefinder we use for obstacle detection cannot measure the range to highly specular surfaces, such as water. This is a practical problem because we do not have independent sensing capabilities, such as polarization sensors, that can detect and measure puddles. In order to circumvent this problem, we indicated the presence of water to the system "by hand" by preventing the robot from driving through it. Although we would have preferred to demonstrate fully autonomous experiments, we believe that this problem is due to shortcomings of the current sensing modality, and it does not affect the fundamental performance of our planning and sensing algorithms.

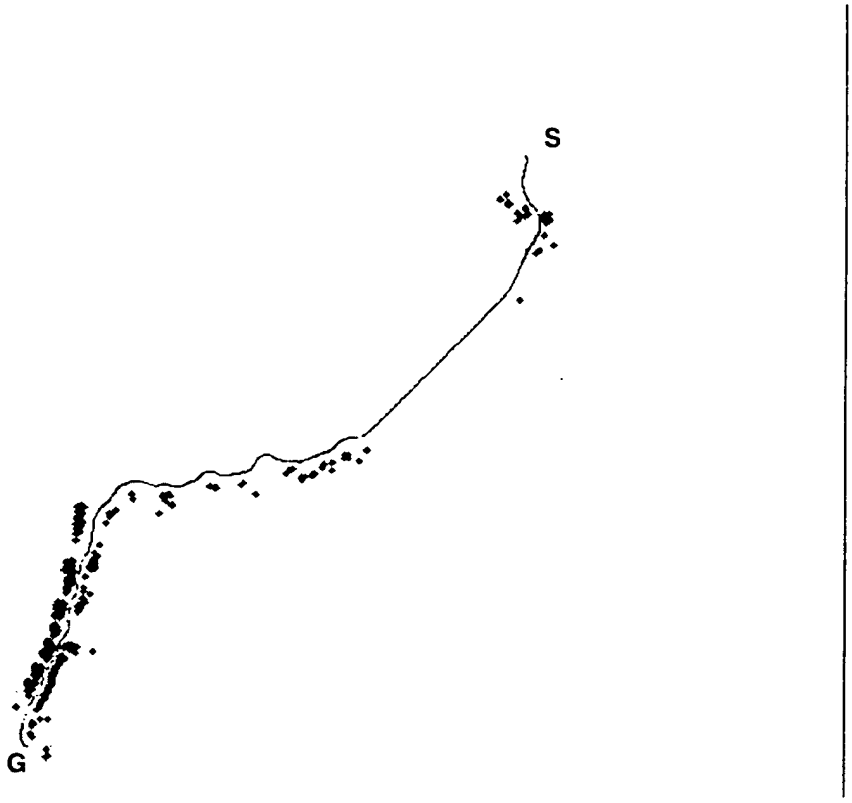


Figure 1: Example of an actual run.

This paper describes the software system for goal acquisition in unknown environments. First, an overview is presented followed by descriptions of the global navigator, local obstacle avoider, and the steering integration system. Second, experimental results from actual robot runs are described. Finally, conclusions and future work are described.

2. The Navigation System

2.1. System Overview

Figure 2 shows a high-level description of the navigation system, which consists of the global navigator, D*, the local navigator, SMARTY, and the steering arbiter, DAMN. The global navigator maintains a coarse-resolution map of the environment, consisting of a Cartesian lattice of grid cells. Each grid cell is labeled untraversable, high-cost, or traversable, depending on whether the cell is known to contain at least one obstacle, is

near to an obstacle, or is free of obstacles, respectively. For purposes of our tests, all cells in the map were initialized to traversable. The global navigator initially plans a trajectory to the goal and sends steering recommendations to the steering arbiter to move the robot toward the goal. As it drives, the local navigator sweeps the terrain in front of the robot for obstacles. The ERIM laser rangefinder is used to measure the three-dimensional shape of the terrain, and the local navigator analyzes the terrain maps to find sloped patches and range discontinuities likely to correspond to obstacles. The local navigator sends steering recommendations to the arbiter to move the robot around these detected obstacles. Additionally, the local navigator sends detected untraversable and traversable cells to the global navigator for processing. The global navigator compares these cells against its map, and if a discrepancy exists (i.e., a traversable cell is now untraversable or vice versa), it plans a new and optimal trajectory to the goal. The key advantage of the global navigator is that it can efficiently plan optimal global paths and is able to generate a new path for every batch of cells in a fraction of a second. The global navigator updates its map and sends new steering recommendations to the steering arbiter.

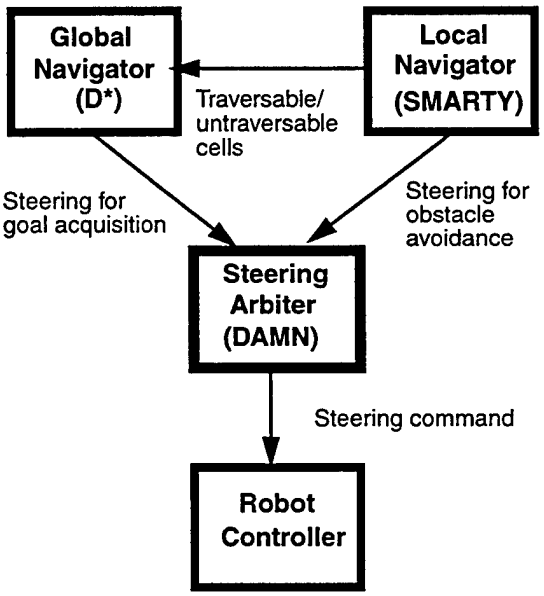


Figure 2: Navigation system diagram.

The steering recommendations sent to the arbiter from the two navigators consist of evaluations for each of a fixed set of constant-curvature arcs (i.e., corresponding to a set of fixed steering angles). The global navigator gives a high rating to steering directions that drive toward the goal, and the local navigator gives a high rating to directions that avoid obstacles. The arbiter combines these recommendations to produce a single steering direction, which is sent to the robot controller.

The rest of this section the global navigator (D*), the local navigator (SMARTY), the steering arbiter (DAMN), and the interfaces between them. Because those modules are described in other papers, we focus here on the modifications that we had to make in order to include them in the cross-country system. In particular, we describe in detail the interfaces between the modules for exchanging steering and map information.

2.2. The D* Algorithm for Optimal Replanning

2.2.1. Overview

The D* algorithm (or *Dynamic* A*) is functionally equivalent to the brute-force replanner (i.e., sound, com-

plete, and optimal), but it is far more efficient. For large environments requiring a million map cells to represent, experimental results indicate that it is over 200 times faster than A* in replanning, thus enabling real-time operation. (See Stentz [29] for a detailed description of the algorithm and the experimental results.)

D* uses a Cartesian grid of eight-connected cells to represent the map. The connections, or arcs, are labelled with positive scalar values indicating the cost of moving between the two cells. Each cell (also called a "state") includes an estimate of the path cost to the goal, and a backpointer to one of its neighbors indicating the direction to the goal.

Like A*, D* maintains an OPEN list of states to expand. Initially, the goal state is placed on the OPEN list with an initial cost of zero. The state with the minimum path cost on the OPEN list is repeatedly expanded, propagating path cost calculations to its neighbors, until the optimal cost is computed to all cells in the map. The robot then begins to move, following the backpointers toward the goal. While driving, the robot scans the terrain with its sensor. If it detects an obstacle where one is not expected, then all optimal paths containing this arc are no longer valid. D* updates the arc cost with a prohibitively large value denoting an obstacle, places the adjoining state on the OPEN list, then repeatedly expands states on the OPEN list to propagate the path cost increase along the invalid paths. The OPEN list states that transmit the cost increase are called RAISE states. As the RAISE states fan out, they come in contact with neighbor states that are able to lower their path costs. These LOWER states are placed on the OPEN list. Through repeated state expansion, the LOWER states reduce path costs and redirect backpointers to compute new optimal paths to the invalidated states.

Conversely, if the robot's sensor detects the absence of an obstacle where one is expected, then a new optimal path may exist from the robot to the goal (i.e., through the "missing" obstacle). D* updates the arc cost with a small value denoting empty space, places the adjoining state on the OPEN list as a LOWER state, then repeatedly expands states to compute new optimal paths wherever possible. In either case, D* determines how far the cost propagation must proceed until a new optimal path is computed to the robot or it is decided that the old one is still optimal. Once this determination has been made, the robot is free to continue moving optimally to the goal, scanning the terrain for obstacles.

For example, Figure 3 shows a 450 x 450 cell simulated environment cluttered with grey obstacles. The black curve shows the optimal path to the goal, assuming all of the obstacles are known a priori. This traverse is known as omniscient optimal, since the robot has complete information about the environment before it begins moving. Figure 4 shows planning in the same environment where none of the obstacles are stored in the map a priori. This map is known as optimistic, since the robot assumes no obstacles exist unless they are detected by its 15-cell radial sensor. This traverse is nearly two times longer than omniscient optimal; however, it is still optimal given the initial map and the sensor information as it was acquired.

2.2.2. Implementation Considerations

A number of modifications were made to D* to adapt it to an actual robot system. The system diagram is shown in Figure 5. As the robot drives toward the goal, its laser rangefinder scans the terrain in front of the robot. The SMARTY local navigator processes this sensor data to find obstacles and sends the (x,y) locations of detected obstacles (untraversable cells) to D* at regular intervals, using the TCX [6] message passing system. Additionally, SMARTY sends (x,y) locations of cells known to be devoid of obstacles (traversable cells). Since the D* map is used to represent a global area, its grid cells are of coarser resolution than SMARTY's (i.e., 1 meter versus 0.4 meter). D* keeps track of the locations of obstacles within each of its grid cells, adding or deleting obstacles as needed based on the data from SMARTY. If an obstacle is added to a previously empty map cell or all of the obstacles are deleted from a previously obstructed cell, then this constitutes a significant event within D* since a traversable cell becomes an untraversable cell or an untraversable cell becomes a traversable cell, respectively.

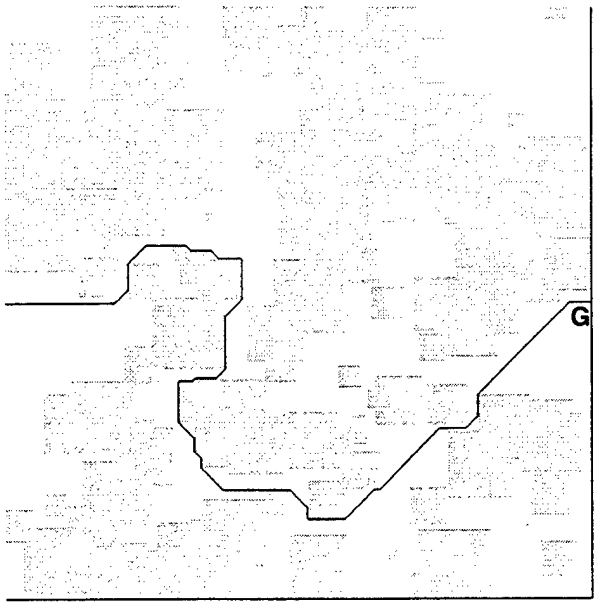


Figure 3: Planning with a complete map.

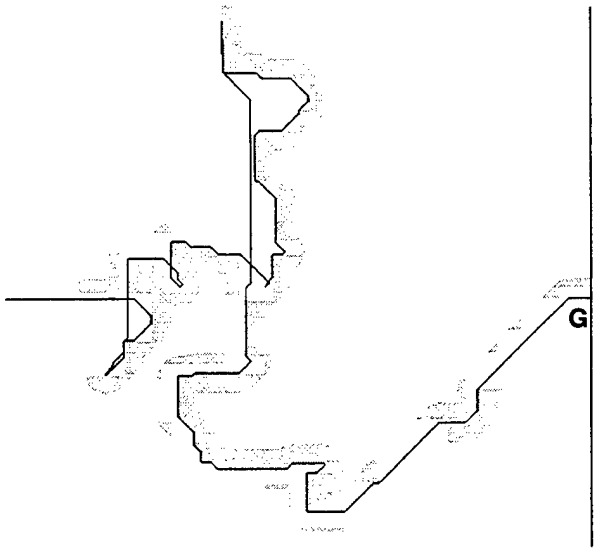


Figure 4: Planning with an optimistic map cell expansion from SMARTY data.

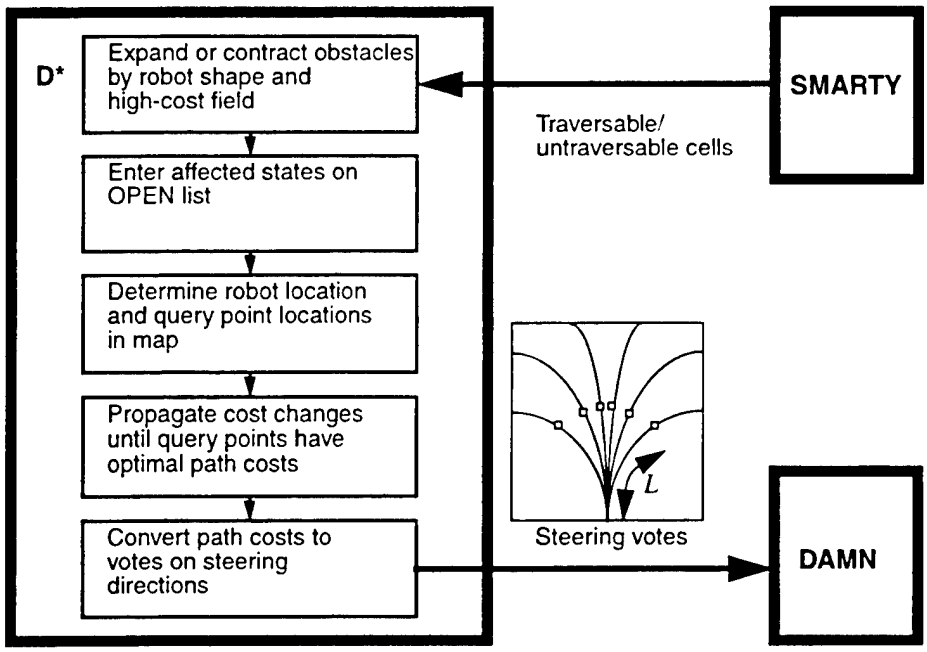


Figure 5: D* overview.

Since D* does not explicitly model the shape of the robot (i.e., it is assumed to be a point), the new untraversable cells are expanded by half the width of the robot to approximate a configuration space (C-space) obstacle. All map cells within a 2-meter radius are classified as untraversable. Note that the longer dimension (i.e., length) of the robot is not used in the expansion. The length must be modeled to detect possible bumper collisions when driving forward or backward; however, D* is not intended to perform local obstacle avoidance. Instead, it is concerned with detecting channels between obstacles that are wide enough to admit the robot if it can get properly oriented. This heuristic breaks down in environments where the channels between obstacles are so constricted and convoluted that the robot's length or minimum turning radius prevents motion through them without collision. In such environments, planning with nonholonomic constraints in a full three-dimensional configuration space is required, but in our experiments we did not encounter the need for it.

The C-space expansion provides some buffering to keep the robot away from obstacles. We found that additional buffering in the form of a high-cost field leads to better performance. The idea is to penalize the robot cost-wise for passing too close to an obstacle, causing the robot to approach obstacles only when open space is unavailable. For example, an area cluttered with trees may be navigable, but a route around the trees would be less risky and therefore preferable. For each true untraversable cell (i.e., those containing obstacles from SMARTY, not those created from the C-space expansion), all traversable cells within a radius of 8 meters are classified as high-cost. The cost of traversing a high-cost cell is five times that of a traversable cell.

The selection of these parameters is a compromise between the degree to which we are willing to take risks by driving through narrow areas misclassified as traversable, and the degree to which we are willing to accept longer routes avoiding areas that are actually open. The degree of acceptable risk is related to the reliability of the sensor processing. That is, the buffer need not be large with a high cost if the sensor classifies cells as traversable with high confidence. We have found empirically that setting the size and cost of the buffer to 8 and 5, respectively, is a good compromise. Therefore, if a channel between two obstacles requires the robot to drive through 10 high-cost cells, it would choose a longer, alternate route passing through up to 50 additional, traversable cells.

When an untraversable cell is changed to traversable, all of the corresponding untraversable and high-cost cells

created during the C-space expansion are changed to traversable. Every time a cell changes classification (i.e., among traversable, untraversable, and high-cost), it is placed on the OPEN list for future consideration. The cost changes are propagated as needed to produce steering recommendations for the DAMN arbiter.

2.2.3. Steering Arc Evaluation for DAMN

Every 500 msec, D* sends steering recommendations to DAMN. D* first checks the robot position and then computes the endpoints of a static set of 51 steering arcs, linearly distributed in arc curvature from -0.125 meter^{-1} to +0.125 meter^{-1}. The points are computed to be at constant distance L from the current robot position along the arcs. L is currently fixed at 10 meters for a speed of about 2 m/sec. D* converts the points to absolute locations in its internal map. It then expands states on its OPEN list until it has computed an optimal path to every point in the list. The cost of the optimal path from each point to the goal is converted to a vote between -1 and +1, where +1 is a strong recommendation and -1 is a veto, and is sent to the arbiter. If a point happens to fall in an untraversable cell, a cell unreachable from the goal, or a cell not in D*'s map, it is assigned a vote of -1. Otherwise, if c_{min} and c_{max} are the minimum and maximum values of the costs in the current list of points, the vote v for an arc is derived from the cost c of the corresponding point in the following way:

$$v = \frac{c_{max} - c}{c_{max} - c_{min}} \quad if\ c_{max} \neq c_{min}$$

$$v = 0 \quad if\ c_{max} = c_{min}$$

This simple formula ensures that arcs going through obstacles are inhibited and that the preferred direction goes through the point of minimum cost. The vote for c_{max} is set to 0 instead of -1 because a high cost means that the arc is less desirable but should not be inhibited.

Because D* does generate enough information to inhibit arcs going through obstacles, one could be tempted to eliminate the obstacle avoidance behavior from SMARTY and to use D* as the sole source of driving commands. Although it is quite possible to configure the system to run in this mode, the result would be poor performance for at least three reasons. First, D*'s map is lower resolution than the map used internally by SMARTY (1 meter vs. 0.4 meter in the current implementation). As a result, D* cannot control the fine motion of the robot. Second, SMARTY typically generates commands faster than D* can update its map and propagate costs, thus ensuring lower latency. Third, SMARTY evaluates the distances between the arcs and all the obstacle cells in the map, whereas D* evaluates only the cost of a single point on the arc. In addition to these practical considerations, it is generally ill-advised to mix local reflexive behavior such as obstacle avoidance and global behaviors such as path planning in a single module.

Finally, we note that the interface between D* and DAMN is actually implemented as a separate module connected through TCX. The module generates steering requests to D* every 500 ms and converts costs to votes and passes them to DAMN. We chose the distributed approach because it isolates D* cleanly from the details of the driving system and because the additional computation and scheduling time would slow down D*'s main expansion loop. Because the query points sent to D* are expressed with respect to the robot, they do not change once L is fixed. We use this property to reduce the message traffic by first sending an initialization message to D*, which contains the list of all the points with respect to the robot. After initialization, the interface simply sends the current robot position to D*, which converts the points to its coordinate system according to this position. In this approach, a request from the interface module consists of a short message of three numbers: x, y, and heading of the robot.

2.3. The SMARTY System for Local Navigation

2.3.1. Overview

SMARTY is a system for controlling a robot based on input from a range sensor. Figure 6 shows SMARTY's basic organization. The range data processing component takes a stream of range pixels as input and outputs untraversable locations with respect to the robot position at the time the data was acquired. The local map manager receives the lists of untraversable cells as soon as they are detected by the range processing portion and maintains their location with respect to current robot position.

The local map manager sends the entire list of untraversable cells to the steering arc evaluation module at regular intervals. The local map is an array of cells with a simpler structure than the grid used in the range data processing component. Local map cells contain only a binary flag indicating whether the cell is traversable; if it is not, the cell also contains the coordinates of a 3-D point inside the obstacle. The positions of the untraversable cells in the local map are updated at regular intervals, currently 100 ms, according to robot motion.

The arc evaluator computes safety scores for a fixed set of arcs based on the relative positions of untraversable cells with respect to the current robot position and sends them to the DAMN steering arbiter. These three parts are implemented as a single Unix process in which the range data is processed continuously as fast as it is acquired, and the arc evaluator is activated at regular time intervals. Like D*, SMARTY communicates with external modules through the TCX messaging system. We briefly describe the SMARTY components in the next paragraphs and conclude this section with a detailed description of the interface between D* and SMARTY.

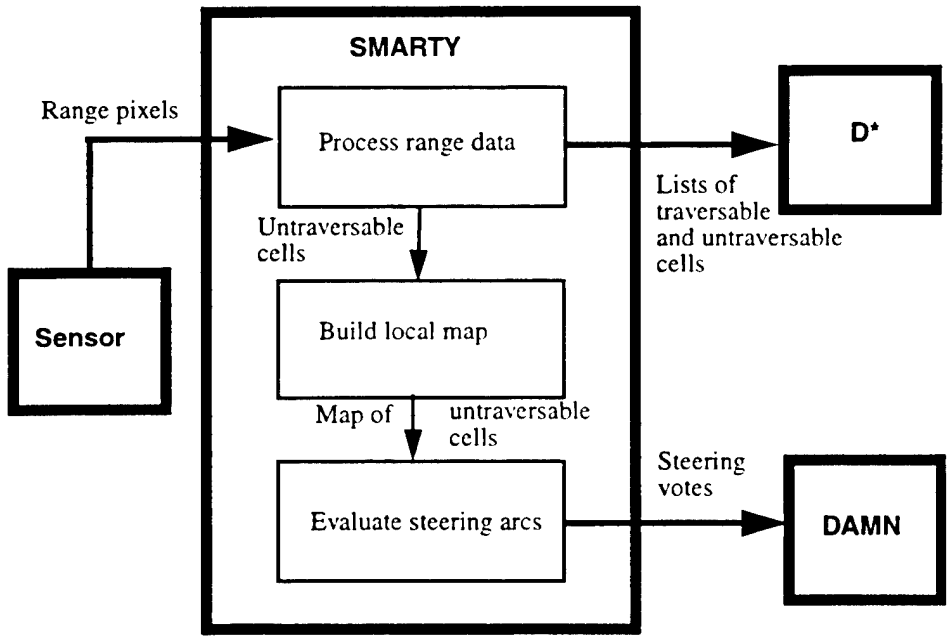


Figure 6: SMARTY overview.

2.3.2. Steering Arc Evaluation for DAMN

The data in the local map is used for generating admissible steering commands. We give here only a brief description of the approach and refer the reader to Keirsey [13] and Langer [17] for a detailed description of

the planning architecture. Each steering arc is evaluated by computing the distance between every untraversable cell in the local map and the arc. An arc receives a vote of -1 if it intersects an untraversable cell; if not, it receives a vote varying monotonically between -1 and +1 with the distance to the nearest untraversable cell. After the vote for each individual arc is computed, the entire array of votes is sent to an arbiter module [13], which generates an actual steering command that is sent to the robot.

Figure 7 shows an example of arc evaluation. Figure 7(a) shows a typical local map. In this display, the untraversble cells are shown as small squares and the position of the vehicle is shown as a rectangular icon. Figure 7(b) shows the distribution of votes for this local map. The distribution of votes ranges from a minimum turning radius of -8 meters to a maximum of +8 meters. The curve of Figure 7(b) shows the maximum votes are for moderate right turns of the robot and are close to -1 for left and right turns.

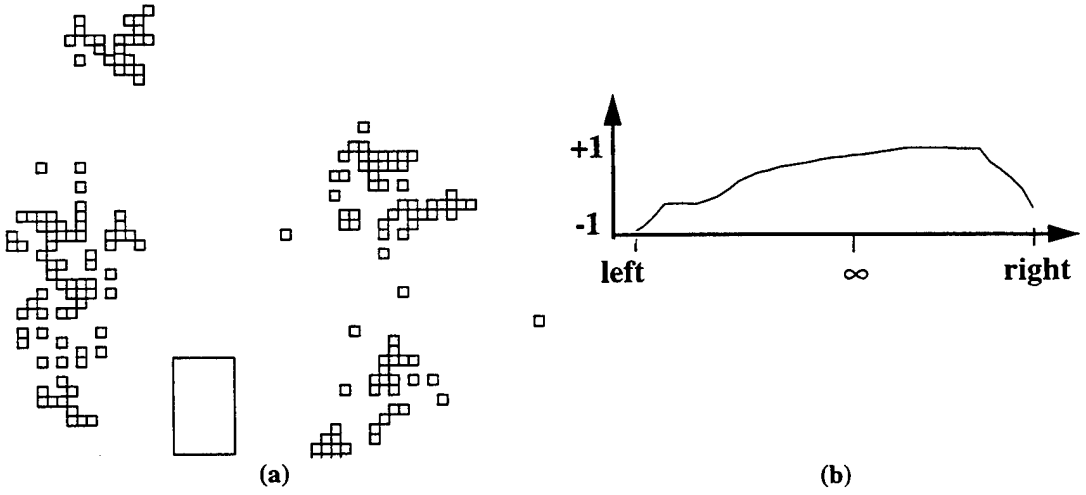


Figure 7: (a) Local map; (b) Corresponding distribution of votes.

2.3.3. Cell Transmission to D*

The system described so far is one of our conventional autonomous driving systems [17]. In order to use SMARTY in conjunction with the D* module, we added a direct link between D* and SMARTY (see Figure 2), because D* needs to update its internal map based on the information extracted from the range images. In theory, SMARTY should send lists of traversable and untraversable cells found in the current batch of range pixels to D* as soon as new data is available. In practice, however, this causes D* to receive data faster than it can process it, due to the overhead in sending and receiving messages.

In order to avoid this problem, the lists of traversable and untraversable cells are buffered instead of being sent to D* as soon as they are computed. In addition, the position of the robot at the time the data used for computing the current cells was acquired is also buffered. The robot position is sent to D* along with the lists of cells. The position information is necessary to enable D* to convert the robot-centered cell locations to cell locations in its internal global map. After a new line of range data is processed, SMARTY flushes the buffer if either of two conditions is met:

- Enough time has elapsed since the last message to D*. The time interval between messages is 500 ms. This value is set empirically for the hardware configuration currently used.
- The position of the robot at the time the most recent sensor data was acquired is different from the position of the robot at the time the data in the buffer was acquired.

The first condition ensures that messages are not generated at a frequency too high for D* to process them. The second condition is necessary because the messages sent to D* include a single robot position so that they cannot accommodate lists of cells acquired from two different robot positions. This message protocol provides a good compromise between the need to send up-to-date information to D* and the need to limit the number of

messages to D*.

2.4. The DAMN System for Steering Arbitration

Our navigation system includes specialized modules, such as obstacle detection (SMARTY) and path planning (D*). Each of the modules has its own view of the best action for the robot to take at every step. We use the DAMN architecture for combining recommendations from different modules and for issuing actual driving commands to the robot controller [17][23][26].

In the system described here, each module is a separate Unix process that communicates with the arbiter through the TCX communication system. Because they may have very different cycle times, the modules operate asynchronously. The arbiter sends commands to the controller at regular intervals, currently 100 ms, and updates the list of combined votes whenever new votes are received from the other modules.

Because the DAMN arbiter does not need to know the semantics of the modules from which it combines votes, it is very general and has been used in a number of systems with different configurations of modules [30]. We concentrate here on the configuration of our navigation system as illustrated in Figure 2. The arbiter receives votes from two modules, D* and SMARTY. The global navigator, D*, sends votes based on its global map and the goal location. The local navigator, SMARTY, sends votes based on the data extracted from range images. The former generates driving recommendations based on global path constraints, while the latter generates recommendations based on a detailed description of the local terrain. Module weights of 0.9 and 0.1 were used for SMARTY and D*, respectively. This selection has the effect of favoring obstacle avoidance over goal acquisition, since it is more important to miss obstacles than to stay on course to the goal.

The current version of DAMN allows for forward motion, but it does not evaluate steering directions for reverse driving. Of course, this is not a problem for on-road navigation systems or for systems that use sufficient a priori knowledge of the environment. In our case, however, it is entirely possible that the only way for the robot to reach the goal is to drive in reverse out of a cul-de-sac. This capability was not yet added to DAMN at the time of this writing so that reverse driving had to be simulated by manually driving the robot out of occasional cul-de-sacs. We clearly indicate such occurrences in the examples given in the next section.

3. Experimental Results

Two of the trial runs that illustrate different aspects of the system are examined in this section. The system was run at a local test site called the Slag Heap, located about 10 minutes from campus. The Slag Heap consists of a large open area of flat terrain bounded by a berm on one side and a large plateau on the other. The obstacles consist of sparse mounds of slag in the interior of the flat area and small trees, bushes, rocks, and debris around its edges. An access road skirts the perimeter of the area. An aerial view of the test site is shown in Figure 8. The dimensions of this area are approximately 800 x 1,000 meters.

3.1. Goal Acquisition with Backtracking

For both trials, an optimistic map was used (i.e., all cells are traversable). S_1 and G_1 mark the start and goal locations for the first trial. S_1 was located in the open area, and G_1 was located on the access road behind the large plateau. These points were chosen so that backtracking would be required to circumnavigate the plateau. Data from the trial at a number of points are shown in Figure 9 through Figure 14. Each of these figures consists of four parts: (a) the robot's trajectory superimposed on D*'s map; (b) the ERIM laser rangefinder image at a selected point along the trajectory; (c) SMARTY's local obstacle map at this same point; and (d) the votes from SMARTY, D*, and DAMN at this same point.

In Figure 9(a), the robot's trajectory is depicted by the black curve. The small rectangle near the end of the

curve is the "selected point" from which the data for subfigures (b), (c), and (d) was taken. The C-space obstacles are shown in dark grey and the high-cost buffer cells in light grey. In Figure 9(b), grey scales encode the range values for the laser rangefinder, with dark grey values near the sensor and light grey values farther away. In Figure 9(c), the local map is shown for the area around the robot. The obstacle cells are shown as squares. The rectangular icon at the bottom of the display shows the position of the robot. A 20-meter local map was used to generate this display. In Figure 9(d), the steering votes for each module are shown, ranging from -1 to +1.

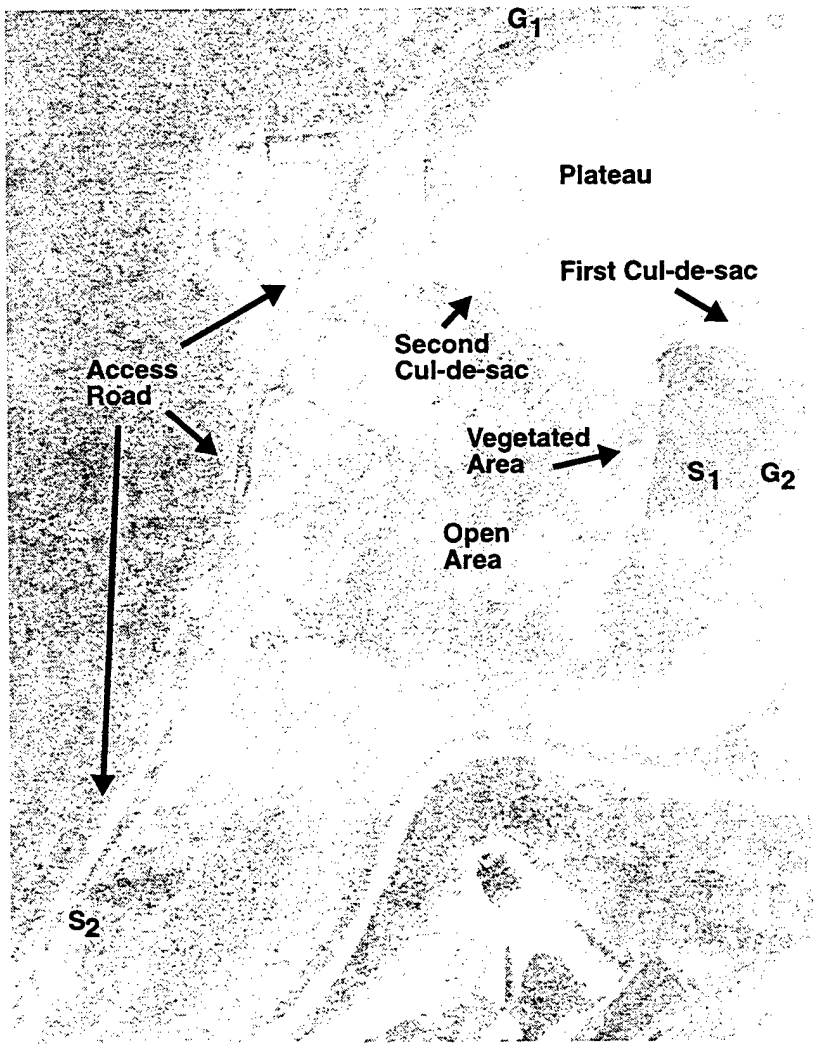


Figure 8: Aerial view of slag heap test site.

Figure 9 shows the first portion of the trial. The robot began pointing away from the goal, so it immediately turned around and headed directly toward it. The robot encountered a large obstruction, initially turned to the left, then looped around the obstacle to the right and drove into a cul-de-sac. At the selected point, SMARTY voted to turn right to avoid the boundary of the cul-de-sac, and D* voted in a similar fashion in order to loop back and explore the other side of the cul-de-sac.

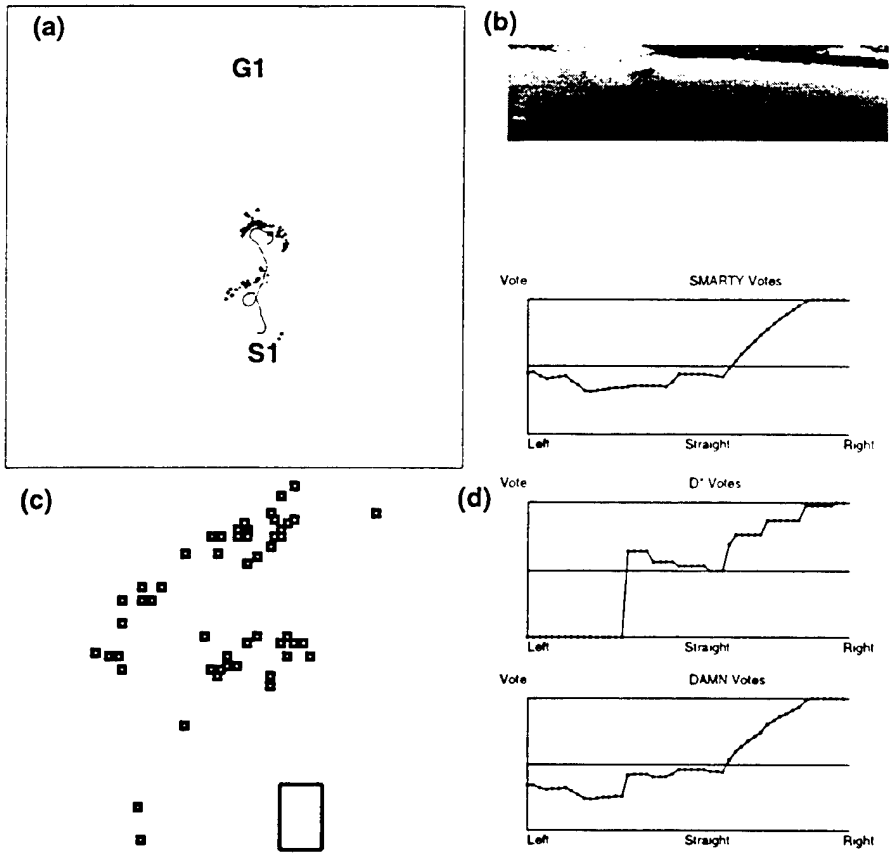


Figure 9: Driving into the cul-de-sac.

Figure 10 shows the robot driving side-to-side, discovering the bounds of the cul-de-sac with its sensor. It appears that, at the selected point, D* preferred to venture into the high-cost buffer rather than backtrack out of the cul-de-sac; it was overruled by SMARTY's votes to avoid the cluttered area altogether. Since D* considers the cost to the goal only from the ends of the steering arcs, it relies on SMARTY to steer clear of obstacles coincident with, or near to, the arcs themselves.

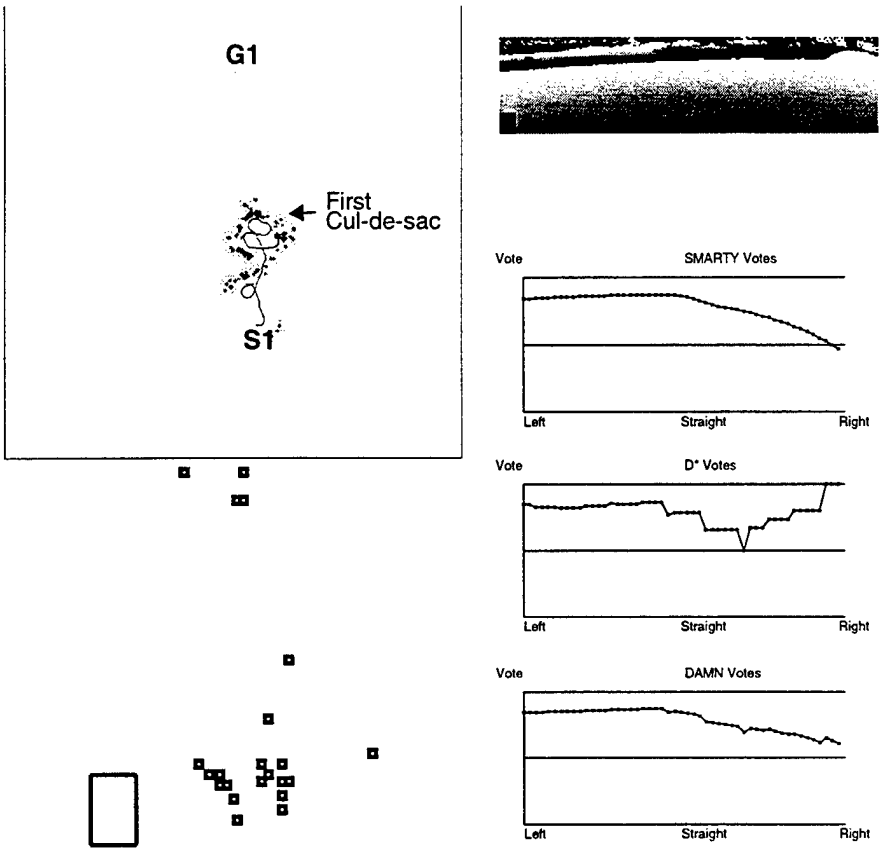


Figure 10: Discovering the cul-de-sac.

Once the robot discovered that the "route" was obstructed, it backtracked out of the cul-de-sac as shown in Figure 11. After exiting, the robot looped back in an attempt to drive through perceived gaps in the surrounding berm. Note that at the selected point, D* voted to turn right and head back toward a gap.

After looping back, the robot closed the first gap with data from the laser rangefinder, and SMARTY deflected the robot away from the second due to other obstacles in the vicinity (see Figure 12).

In Figure 13, the robot moved into an area densely populated with small trees before driving out to the left. In these types of areas, the robot was driven predominantly by SMARTY, since obstacle avoidance takes precedence over the goal seeking behavior. After emerging from the area, D* guided the robot around the perimeter of the vegetation and into another small cul-de-sac. As was the case with the first cul-de-sac, the limited field of view of the ERIM sensor precluded the possibility of detecting the cul-de-sac before entry and avoiding it altogether.

This time the cul-de-sac was too small for the robot to escape without driving in reverse. Since the NAVLAB II is currently unable to do this automatically, the robot was manually driven in reverse until it exited the cul-de-sac. Note that, at the selected point, SMARTY detected obstacles in all directions and consequently voted against all steering arcs. D* favored backing out, but since such a move was not possible autonomously, it voted for the next best options: either a hard left or hard right turn.

After backing out, the robot was placed under autonomous control once again (see Figure 14). It drove around the perimeter of the large plateau, found an entry point to the road behind the plateau, and then drove along the access road until it reached the goal.

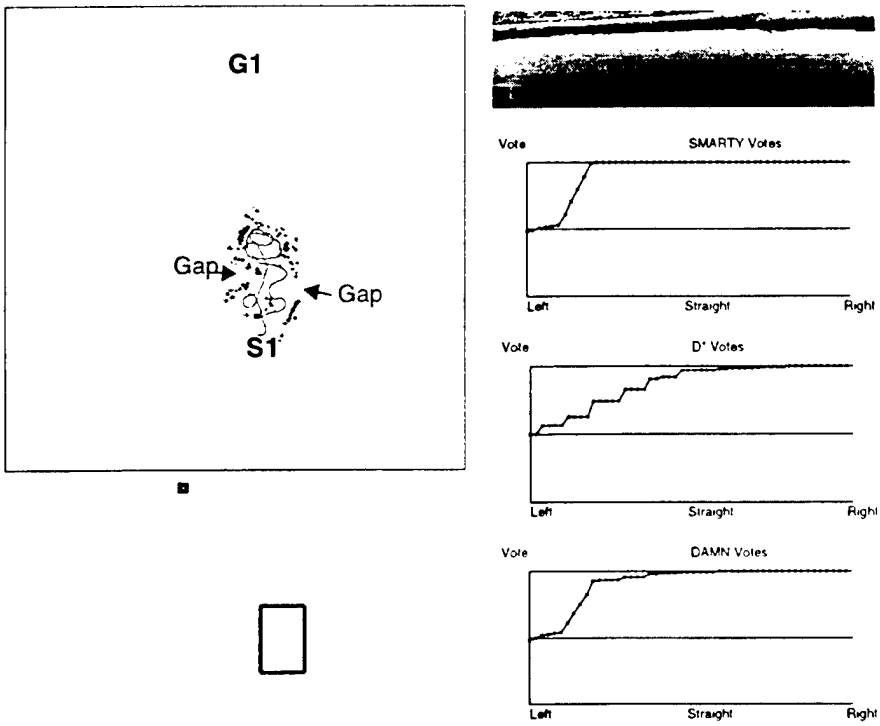


Figure 11: Exiting the cul-de-sac.

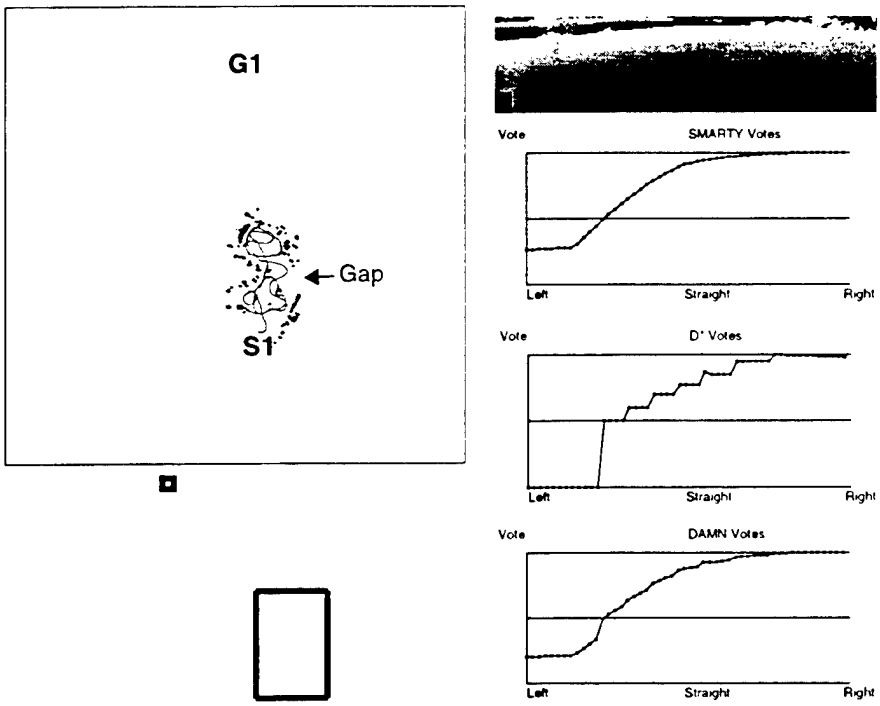


Figure 12: Looping back toward the "gap".

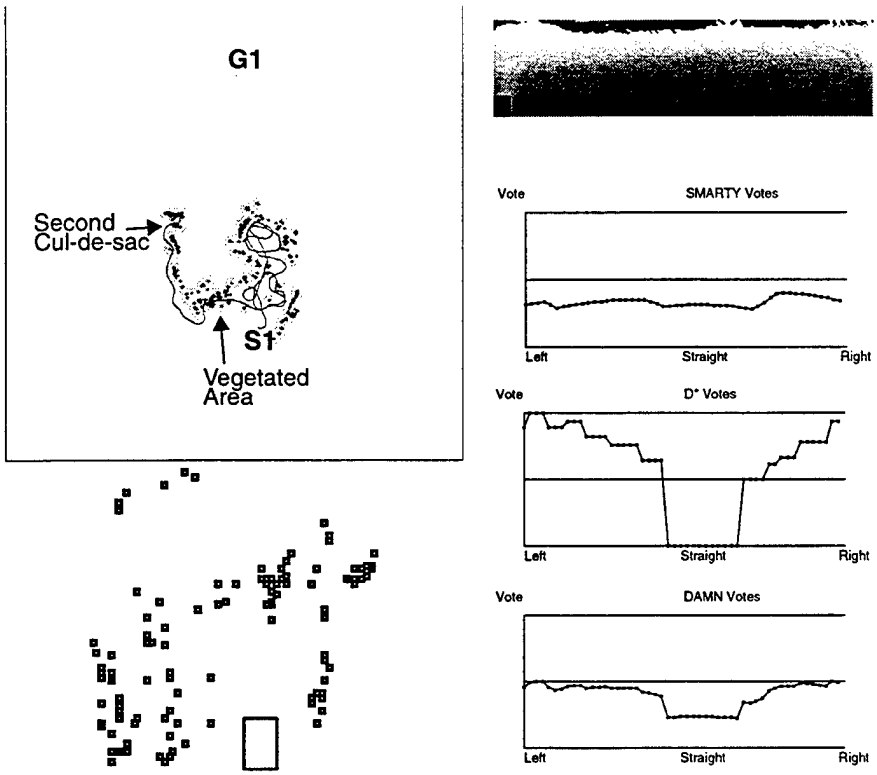


Figure 13: Driving through and around the vegetated area.

The total path length for the trial was 1410 meters. At six points during the trial, we manually intervened to steer the robot. Half of these interventions were to drive the robot in reverse, and the other half were steering deflections to avoid water or mud that was too difficult to detect with a laser rangefinder. D* sent steering recommendations to the DAMN arbiter every 500 ms. A total of 2952 sets of steering recommendations were sent. Since each set consists of 51 steering arcs, a total of 150,552 global paths to the goal were computed during the trial. SMARTY sent 6119 messages to D* containing a total of 1,851,381 terrain cell classifications. The radius of C-space expansion was 2 meters, and the radius of each high-cost buffer was 8 meters. A high-cost cell was five times more expensive to traverse than a traversable cell.

The number of cell classifications was large since each terrain cell is likely to be seen more than once, and each occurrence is transmitted to D*. It is also important to note that the classification for many terrain cells changed repeatedly from one sensor reading to the next. This effect was due in part to sensor noise and in part to the fact that the classification of a given cell improves in accuracy as the robot draws nearer and the sensor gets a better view.

Note that the high-cost buffer was essential to complete the boundaries of the cul-de-sacs, plateau, and roads. Without it, the robot would need to loop around many more times for more sensor data in the first cul-de-sac before D* became convinced the route was obstructed. We observed this behavior in actual experiments. In some cases, the robot had to loop up to ten times inside the cul-de-sacs in order to map the entire boundary of the region. Although this behavior is entirely justified from a planning perspective, since the robot needs to explore an entire area before determining it must backtrack out of it, it is clearly unacceptable from a practical standpoint. Better sensing is the proper solution.

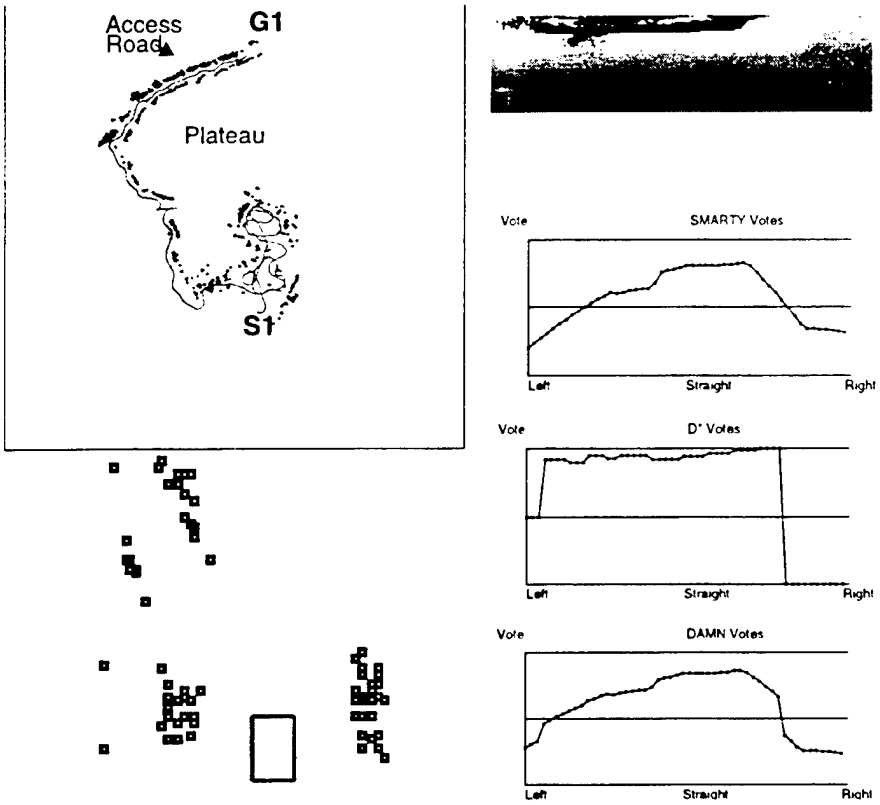


Figure 14: Finding the access road that leads to the goal.

3.2. Using Map Data from Prior Trials

In Figure 8, S_2 and G_2 mark the start and goal locations, respectively, for the second trial. The start was chosen to be at the entrance to the Slag Heap on the access road, and the goal was at the opposite end of the flat, open area. The objective of this trial was not to illustrate a difficult path to the goal; instead, it was to illustrate the effects of multiple runs over the same area. In Figure 15, the robot drove down the access road and across the open area to the goal. At the goal point, the robot was taken out of automatic control and driven manually to another portion of the access road (see Figure 16). During the manually driven segment, the software continued to run; thus, the map was updated with obstacles detected by the sensor.

The robot was placed under automatic control again, and it drove down a segment of the access road until it found an entry point into the open area. It then proceeded to drive across the open area to the goal, avoiding a number of obstacles along the way (see Figure 17).

The robot was driven manually once again to a new start point (see Figure 18) and placed back under automatic control. It then drove to the goal for a third time (see Figure 19). Note that, in its third path to the goal, the robot used map data that was constructed during the previous traverses. As shown in the figure, the robot positioned itself to pass between two obstacles before its sensor was close enough to spot them.

The length of the robot's path for this trial was 1664 meters, including both automatically and manually driven segments. D* sent a total of 3168 sets of steering recommendations to DAMN; thus, a total of 161,568 global paths were computed. SMARTY sent 6581 messages to D* with a total of 1,601,161 cell classifications.

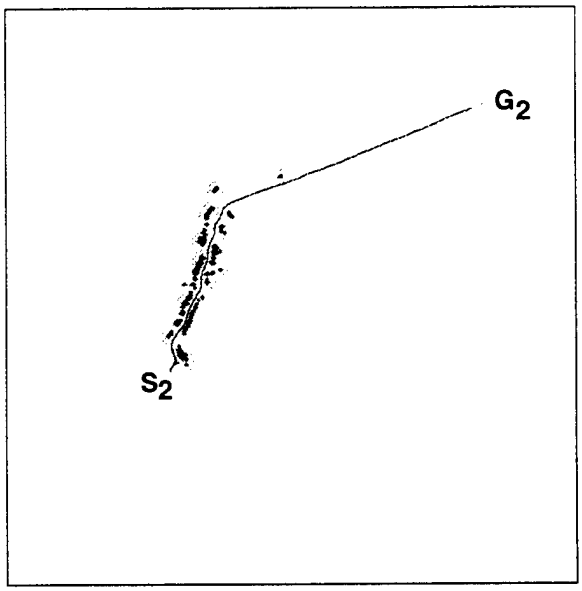


Figure 15: Driving the initial path to the goal.

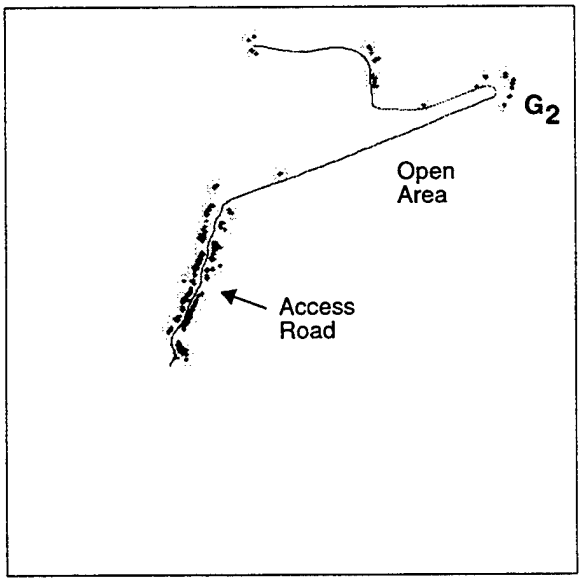


Figure 16: Manually driving to a new Start point.

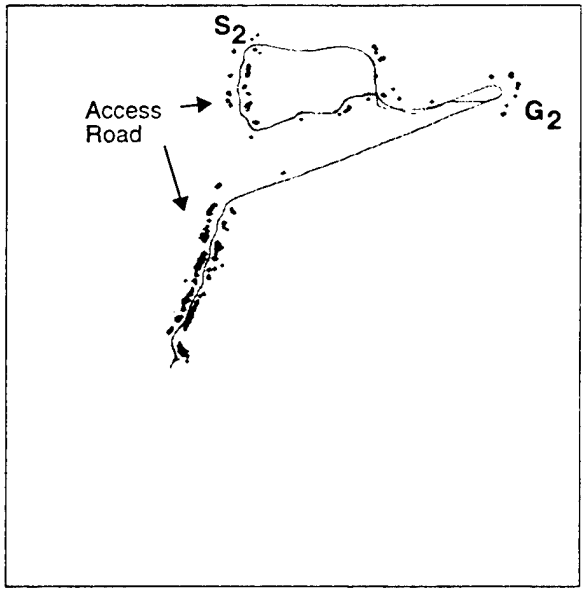


Figure 17: Finding the goal for a second time.

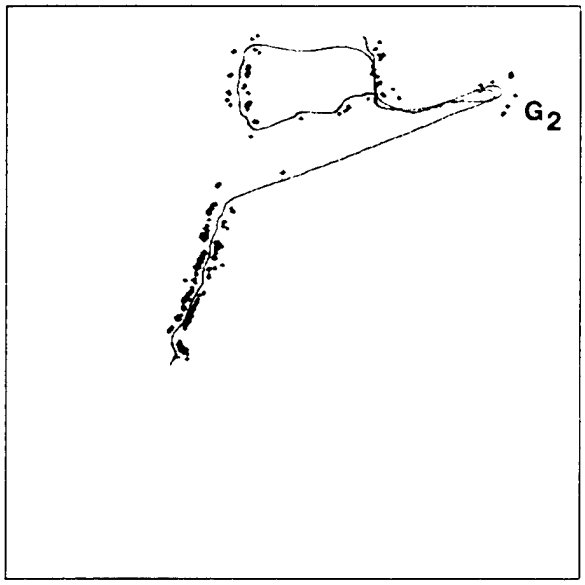


Figure 18: Driving manually to a third start point.

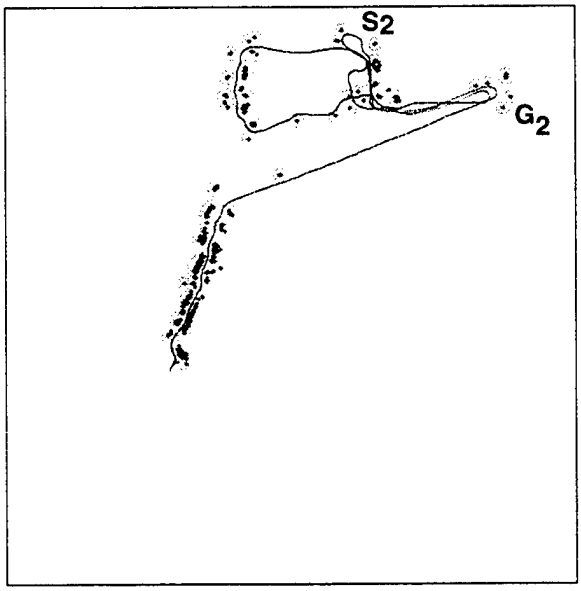


Figure 19: Driving to the goal using prior map data.

4. Conclusions

4.1. Summary

This paper describes a complete navigation system for goal acquisition in unknown environments. The system uses all available prior map data to plan a route to the goal, and then begins to follow that route using its laser rangefinder to examine the terrain in front of the robot for obstacles. If a discrepancy is discovered between the sensor data and the map, the map is updated and a new, optimal path is planned to the goal. Because of the efficiency of D*, this new path can be generated in a fraction of a second. Both the global navigator (D*) and the local navigator (SMARTY) send steering recommendations to the steering arbiter (DAMN). Because obstacle avoidance takes precedence over goal acquisition, the votes from SMARTY are weighted more heavily than those from D*. Thus, in areas dense with obstacles, the robot is driven primarily by SMARTY, while in open areas, it is primarily driven by D*. It was found that the high-cost buffers around obstacles were essential to fill in gaps between obstacles and preclude repeated sensing of the same area. It was also discovered that the two-dimensional approximation of the robot's three-dimensional configuration space was sufficient for the experiments conducted.

To our knowledge, this system is the first to demonstrate efficient goal acquisition and obstacle avoidance on a real robot operating in an unstructured, outdoor environment.

4.2. Future Work

In the near-term, a number of improvements will be made to minimize unnecessary processing and increase overall system speed. An enormous number of terrain cell classifications are transmitted from SMARTY to D*. Some of these classifications are erroneous due to noise or less-than-ideal viewing conditions. Noise filtering and verification of the classifications across sensor images would increase confidence in the data and reduce communications traffic. Currently, the cell classifications received by D* are processed sequentially to create the C-space expansions and high-cost buffers. This approach is highly inefficient given typical cluster-

ing of obstacles, and the additional computational burden resulted in D* sending less-than-optimal steering recommendations in cluttered areas in order to meet timing deadlines. Processing the obstacles in batch mode using the grassfire transform for expansion should greatly reduce this overhead. Furthermore, we will develop a more systematic approach to the scheduling of the interactions between SMARTY and the other modules, D* and arbiter. Currently, the frequency at which SMARTY sends information to the other modules is determined empirically for a typical robot speed. We will develop an algorithm that relates all the system parameters, such as sensor field of view or robot speed, to the communication frequency. This last improvement will involve moving the modules to a real-time operating system in order to guarantee repeatable performance.

In the far-term, we will extend D* so that it dynamically allocates new map storage as needed rather than requiring the map to be pre-allocated to a fixed size. Furthermore, we will add the capability in D* to reason about robot maneuvers (coupled forward-backward motion) in order to handle very cluttered areas that require such complicated motion. We will include a mechanism for speed control in SMARTY in addition to the existing mechanism for steering control. Speed control involves reasoning about the distribution of map cells with respect to the robot and issuing recommendations for speed settings such that the robot slows down in cluttered environments. The speed recommendations may be encoded very much like the steering recommendations: a set of votes for speed values between 0 and a pre-set maximum speed. Additionally, we will improve the performance of SMARTY in terms of speed and maximum range in order to support higher robot speed. This will be achieved mostly by using better sensors, such as single-line laser scanners or passive stereo.

Bibliography

[1] Bhatt, R., Venetsky, L., Gaw, D., Lowing, D., Meystel, A. A Real-Time Pilot for an Autonomous Robot. *Proceedings of IEEE Conference on Intelligent Control*. 1987.

[2] Chatila, R., Devy, M. Herrb, M. Perception System and Functions for Autonomous Navigation in a Natural Environment. *Proceedings of CIRFFSS*. 1994.

[3] Daily, M., Harris, J., Keirsey, D., Olin, K., Payton, D., Reiser, K., Rosenblatt, J., Tseng, D., Wong, V., Autonomous Cross Country Navigation with the ALV. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1988.

[4] Dickmanns, E.D., Zapp, A. A Curvature-Based Scheme for Improving Road Vehicle Guidance by Computer Vision. *Proceedings of the SPIE Conference on Mobile Robots*. 1986.

[5] Dunlay, R.T., Morgenthaler, D.G. Obstacle Avoidance on Roadways Using Range Data. *Proceedings of SPIE Conference on Mobile Robots*. 1986.

[6] Fedor, C. *TCX, Task Communications User's Manual. Internal Report*. The Robotics Institute, Carnegie Mellon. 1993.

[7] Feng, D., S., Krogh, B. Implementation of Dynamic Obstacle Avoidance on the CMU Navlab. *Proceedings of IEEE Conference on Systems Engineering*. 1990.

[8] Franke, U., Fritz, H., Mehring, S. Long Distance Driving with the Daimler-Benz Autonomous Vehicle VITA. PROMETHEUS Workshop, Grenoble. 1991.

[9] Gat, E., Slack, M. G., Miller, D. P., Firby, R. J. Path Planning and Execution Monitoring for a Planetary Rover. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1990.

[10] Goto, Y., Stentz, A. Mobile Robot Navigation: The CMU System. *IEEE Expert*, Vol. 2, No. 4. 1987.

[11] Hebert, M. Pixel-Based Range Processing for Autonomous Driving. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1994.

[12] Hebert, M.,Krotkov, E. 3D Measurements from Imaging Laser Radars. *Image and Vision Computing*, Vol. 10, No. 3. 1992.

[13] Keirsey, D.M., Payton, D.W. and Rosenblatt, J.K. Autonomous Navigation in Cross-Country Terrain. In *Proceedings Image Understanding Workshop*. Cambridge, MA. 1988.

[14] Kelly, A. J. *RANGER - An Intelligent Predictive Controller for Unmanned Ground Vehicles.* Technical Report, The Robotics Institute, Carnegie Mellon. 1994.

[15] Kluge, K. *YARF: An Open-Ended Framework for Robot Road Following.* Ph.D. Dissertation, CMU-CS-93-104. School of Computer Science, Carnegie Mellon University. 1993.

[16] Korf, R. E. Real-Time Heuristic Search: First Results. *Proceedings of the Sixth National Conference on Artificial Intelligence.* 1987.

[17] Langer, D., Rosenblatt, J.K., Hebert, M. An Integrated System for Autonomous Off-Road Navigation. *Proceedings of the IEEE International Conference on Robotics and Automation.* 1994.

[18] Latombe, J.-C. *Robot Motion Planning.* Kluwer Academic Publishers. 1991.

[19] Lumelsky, V. J., Stepanov, A. A. Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment. *IEEE Transactions on Automatic Control.* Vol. AC-31, No. 11.1986.

[20] Matthies, L. Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time Implementation. *International Journal of Computer Vision.* Vol. 8, No. 1. 1992.

[21] McTamaney, L.S. Mobile Robots: Real-Time Intelligent Control. *IEEE Expert.* Vol. 2, No. 4. 1987.

[22] Nilsson, N. J. *Principles of Artificial Intelligence.* Tioga Publishing Company. 1980.

[23] Payton, D.W., Rosenblatt, J.K., Keirsey, D.M. Plan Guided Reaction. *IEEE Transactions on Systems Man and Cybernetics.* Vol. 20, No. 6. 1990.

[24] Pirzadeh, A., Snyder, W., "A Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control," Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[25] Pomerleau, D.A., "Efficient Training of Artificial Neural Networks for Autonomous Navigation," Neural Computation, Vol. 3, No. 1, 1991.

[26] Rosenblatt, J.K., Payton, D.W., "A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control," in Proceedings of the IEEE/INNS International Joint Conference on Neural Networks, Washington DC, Vol. 2, pp. 317-324, June, 1989.

[27] Singh, S., Feng, D., Keller, P., Shaffer, G., Shi, W.F., Shin, D.H., West, J., Wu, B.X., "A System for Fast Navigation of Autonomous Vehicles," Robotics Institute Technical Report CMU-RI-TR-91-20, Carnegie Mellon University, 1991.

[28] Stentz, A., Brumitt, B.L., Coulter, R.C., Kelly, A., "An Autonomous System for Cross-Country Navigation," Proceedings of the SPIE Conference on Mobile Robots, 1992.

[29] Stentz, A., "Optimal and Efficient Path Planning for Partially-Known Environments," Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

[30] Thorpe, C., Amidi, O., Gowdy, J., Hebert, M., Pomerleau, D., "Integrating Position Measurement and Image Understanding for Autonomous Vehicle Navigation," Proceedings of the *Workshop on High Precision Navigation*, Springer-Verlag Publisher, 1991.

[31] Wilcox, B., Matthies, L., Gennery, D., Cooper, B., Nguyen, T., Litwin, T., Mishkin, A., Stone, H., "Robotic Vehicles for Planetary Exploration," Proceedings of the IEEE International Conference on Robotics and Automation,1992.